(19)

Direct Incorporation of Uncertainty in Chemical and Environmental Engineering Systems

by

Menner A. Tatang

Submitted to the Department of Chemical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1995

Author .....

Department of Chemical Engineering
October 14, 1994

Certified by .....

Gregory J. McRae
Joseph R. Mares Professor
Thesis Supervisor

Accepted by .....

Robert E. Cohen
Chairman, Departmental Committee on Graduate Students

Direct Incorporation of Uncertainty in Chemical and Environmental Engineering Systems

by

Menner A. Tatang

Submitted to the Department of Chemical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1995

Author .....
Department of Chemical Engineering
October 14, 1994

Certified by .....
Gregory J. McRae
Joseph R. Mares Professor
Thesis Supervisor

Accepted by .....
Robert E. Cohen
Chairman, Departmental Committee on Graduate Students

Direct Incorporation of Uncertainty in Chemical and Environmental Engineering Systems

by
Menner A. Tatang

Submitted to the Department of Chemical Engineering
on October 14, 1994, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Uncertainty always exists in chemical and environmental engineering systems. The critical problem, in practice, is to evaluate the effects of uncertainties on predicted outcomes. This thesis proposes a computationally efficient method which allows parametric uncertainty to be incorporated explicitly in the problem formulation. Key features of the method, termed the deterministic equivalent modeling method (DEMM), include the use of symbolic manipulation and compiler technology, and the ability to solve stochastic models with existing numerical algorithms used for the corresponding deterministic problems. Furthermore, the use of collocation approach in the DEMM environment extends the application area of DEMM to black-box or implicit type models.

A prototype language for this proposed approach has been implemented. Such an implementation serves not only to demonstrate the ease of conducting uncertainty analysis for a wide range of chemical and environmental engineering problems, but also to provide an environment where the approach can be further examined and improved. Examples of models in chemical and environmental engineering systems are considered, and their results suggest both the accuracy of the deterministic equivalent modeling method and the practicality of its prototype language for performing uncertainty analyses.

Thesis Supervisor: Gregory J. McRae
Title: Joseph R. Mares Professor

Acknowledgments

Carrying out a research work and writing it as a doctoral thesis are a unique and colorful experience in one's life. I would like to extend my sincere appreciation and gratitude to my advisor, Prof. Gregory J. McRae, for providing me with such an experience, and whose invaluable guidance and extensive knowledge have fostered an open and supportive atmosphere. I would also like to thank Prof. Robert C. Armstrong, Prof. Paul I. Barton, and Prof. Ignacio E. Grossmann, for their contributions and time spent as members of my thesis committee.

To my office mates: Jonathan Knight, Markus Langner, Gene Lin, Wen Wei Pan, Betty Pun, Timothy Resch, and Peter Wyckoff, thanks for all of those fruitfull discussions in our research group. Special thanks go to Janet Fischer and Elaine E. Aufiero-Peters of Student Office for their friendly assistance and source of information.

I am also indebted to my wife, Suryawati S. Tatang, whose constant love has allowed me to pursue this goal without fear of failure. A warm thank goes to my parents, Effendy A. Tatang and Tjendrawati Tjandra, for willingly taking care many things.

This research was supported in part by the NSF, and the US-EPA. This support is gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis statement

There is no question that uncertainty is always there in real systems. Unfortunately, there are several barriers in performing uncertainty analysis, which include obtaining the information of uncertainty in inputs; selecting a consistent and general uncertainty representation; the dimensionality of uncertain inputs; and the difficulty in implementing the analysis. To cope with such a complexity, we need an efficient methodology which allows direct incorporation of uncertainty in the model formulation. This thesis proposes a computationally efficient methodology termed the deterministic equivalent modeling method (DEMM) for dealing with parametric uncertainty in chemical and environmental engineering systems.

A strong impact of this research work to the chemical engineering community is mostly the reduction of complexity for performing uncertainty analysis to various chemical engineering models. In achieving that impact, several contributions of this thesis are identified:

- Introducing and applying a number of useful concepts for uncertainty representations, such as polynomial chaos expansions, to chemical and environmental engineering problems.
- Developing a consistent and general notion of deterministic equivalent models.
- Developing and applying the probabilistic variational approach to reduce the complexity of stochatic models.
- Emphasizing the practicality of reusing deterministic numerical solvers for solving corresponding stochastic models.
- Developing and applying a prototype language as an environment for carrying out uncertainty analysis.
- Developing and applying the collocation method in the DEMM setting to black-box type models, such as modular-based process simulators.
- Demonstrating the applications of the DEMM to numerous chemical and environmental engineering models.

1.2 Uncertainty analysis and its barriers

In practice, the exact values of numerous quantities entering a model are usually unknown because they can not be precisely measured [81]. Such a problem brings the need to systematically and explicitly address uncertainty in these quantities in order to investigate its effects to the design, operation, and assessment of chemical and environmental engineering systems. Figure 1-1, which is modified and drawn from [5, 16, 32, 96, 109, 152], shows various possible sources of uncertainty in chemical process systems. To address those uncertainties we are required to perform an uncertainty analysis. In general, uncertainty analysis can be defined as a study to compare the importance of the input uncertainties in terms of their relative contributions to uncertainty in the outputs [112]. In other words, there are two elements of uncertainty analysis:

Sensitivity analysis A study which resolves the effect of changes in inputs on model predictions.

Uncertainty propagation The computation of the uncertainty in the model outputs that is induced by the uncertainties in its inputs.

Since in most cases uncertainty propagation may also produce the results of the sensitivity analysis [113], the focus of uncertainty analysis is mainly in the second element. Consequently, the problem of uncertainty analysis can be posed as follows:

> *Given a description of the input uncertainties, how can we obtain a description of uncertainty in the outputs?*

The answer to this question depends on two factors. One is the structure of the model in consideration, and the other is the representations of uncertainty. Three forms of models are considered in this thesis. They include equality, inequality, and optimization models. An optimization model can be seen as a complete problem because it may include both equality and inequality models as its equality and inequality constraints respectively. As an example, an optimally designed chemical plant can be formulated by the following optimization model:

$$
\begin{aligned}
\min_{d \equiv \text{design variables}} \quad & \text{Total cost} \equiv C(d, x, z, \theta) \\
\text{subject to} \quad & \text{Performance equations :} \\
& H(d, x, z, \theta) = 0 \\
& \text{Specifications and capacity constraints :} \\
& G(d, x, z, \theta) \leq 0,
\end{aligned}
$$

where x and z denote vectors of state and control variables respectively, and θ is a set of parameters. Inputs to this model are the values of parameters θ. As stated previously, these inputs or parameters may contain uncertainty. Such a problem belongs to the class of parametric uncertainty models, and this thesis focuses only on that class of uncertainty.

Parametric uncertainty in general implies that one is unsure of the particular value a parameter will assume. It means that a parameter's value can be uncertain if either one of the following two cases exists [98]:

- the parameter is single-valued, deterministic, and constant, but its value is not perfectly known at the design or decision time.

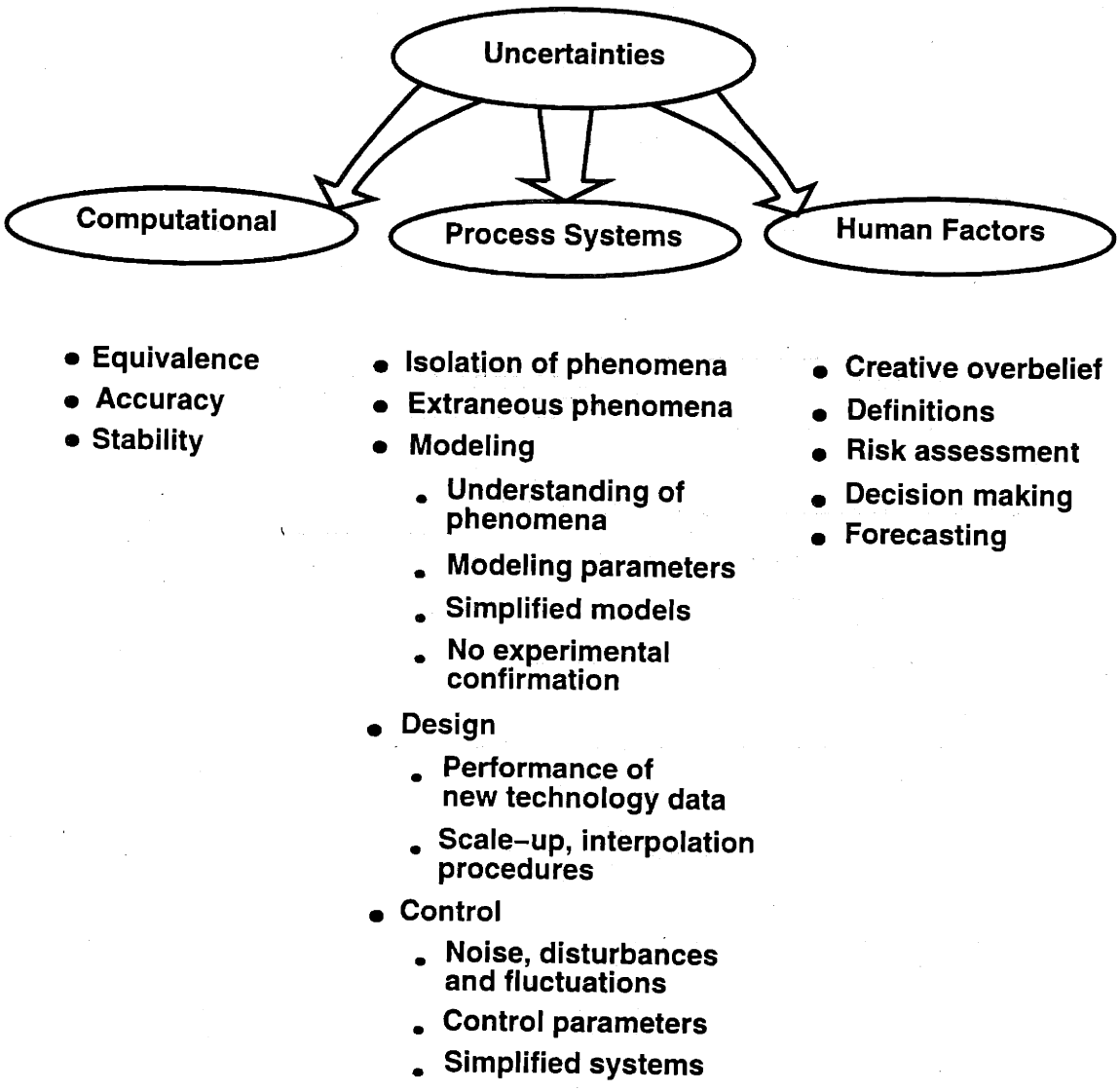


Figure 1-1: Numerous sources of uncertainty in the process system.

- the parameter's value is constantly fluctuating, with a random pattern, or it has imperfectly known variability.

In both cases, parametric uncertainty may be represented by a probability density function. This brings the discussion to the choice of parametric uncertainty representations. In addition to the probabilistic description, there are variety of ways to represent parametric uncertainty including such techniques as interval mathematics and fuzzy set theory. On account of theoretical foundations of probabilistic representation are already mature, this thesis choose to use the probabilistic description. Further discussion on these representations will be given in chapter 3.

In the probabilistic case, the main question of uncertainty analysis can be restated as (see also figure 1-2):

> *Given the joint probability density function of parameters, how can we obtain the joint probability density function of response variables?*

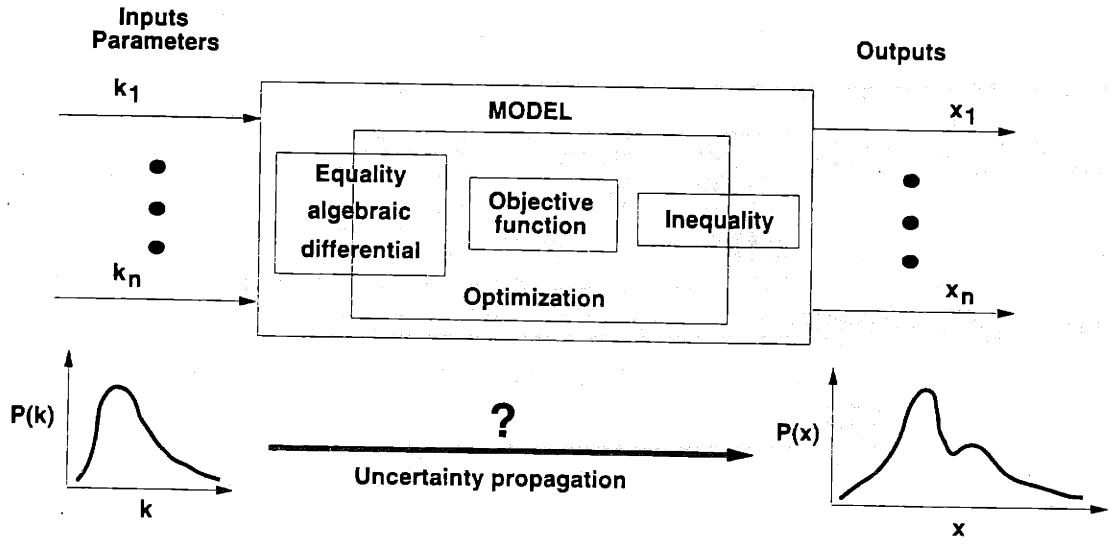


Figure 1-2: Uncertainty propagation is a major problem in uncertainty analysis; the structure of model is represented by a Venn diagram.

There are, in fact, several barriers to answer this question, or to perform uncertainty analysis:

Joint density function of parameters Without having a representative estimate of the probability density function of uncertain parameters, the uncertainty analysis may produce incorrect results. This thesis recommends the use of Bayesian approach [161] or the minimum cross-entropy method [88, 149] to estimate, and to update the density function of uncertain parameters.

Representation of random variables A consistent and general representation of random variables is necessary for dealing with different structures of models. Fortunately, a general type of representations is available, such a representation is known as the polynomial chaos expansion [190].

Curse of dimensionality A major problem in performing uncertainty analysis is the dimensionality of uncertain parameters. As an example, evaluating common metrics such as the mean value of outputs may require a very large number of sampling points in order to develop stable statistics. The calculation of moments of the optimal value of the optimization model posed above, which are obtained through multi-dimensional integration, illustrates the "curse of dimensionality",

$$\begin{aligned} m_i &= E(y^i(\theta)) \\ &= \underbrace{\int \cdots \int}_{p} \{y \equiv min_d C(d,x,z,\theta) | H(d,x,z,\theta) = 0, G(d,x,z,\theta) \leq 0\}^i \times \\ &\quad f_\theta(\theta)\, d\theta_1 \cdots d\theta_p, \end{aligned}$$

where $f_\theta(\theta)$ is the joint density function of a set parameters $\theta = \{\theta_1, \ldots, \theta_p\}$. The error bound for calculating that integral with the trapezoidal rule is $O(N^{-\frac{2}{p}})$ [115]. Specifically, to guarantee a prescribed level of accuracy, say an error that is in absolute value $\leq 10^{-2}$, we must use approximately 10^p nodes. In other words, the required number of nodes increases exponentially with the dimension p. Such a problem is often called as the "curse of dimensionality".

To avoid such a "curse", we can refer to the Monte Carlo methods which promises that the *probabilistic* error bound is independent of the dimension, $O(N^{-\frac{1}{2}})$. From a practical point of view what is needed is a deterministic error bound. This requirement fosters the study of quasi-random sampling points. While the Monte-Carlo method requires the number of nodes $N = O(\frac{1}{\varepsilon^2})$ to obtain the probabilistic error ε, the quasi-Monte Carlo method, such as the Hammersley-Wozniakowski sampling method [193], yields a much better result, giving us the number of nodes needed as the average *deterministic* error ε goes to zero,

$$N = O\left(\frac{1}{\varepsilon} \log\left(\frac{1}{\varepsilon}\right)^{\frac{p-1}{2}}\right).$$

Table 1.1 shows the comparison of the number of nodes needed for the three different methods. Clearly, in the case that model evaluation is expensive, both the Monte Carlo methods and quasi-Monte Carlo methods are also intractable for a reasonable accuracy.

Ease of implementation Implementing Monte Carlo methods is relatively easy, however, as mentioned previously its computation may be prohibitively expensive. Consequently, any alternative methods should have as simple implementation as of the Monte Carlo methods, and also must retain the accuracy of solution. The first requirement suggests the need to use the same deterministic solvers as Monte Carlo methods use for solving the model.

1.3 Objectives

Reducing the complexity, described in the previous list of barriers, in any activities involving uncertainty analysis becomes the central objective of this research. In achieving that objective, several key elements of this work are identified:

Error bound	Dimension	Number of nodes		
		Trapezoidal	Monte Carlo	quasi-Monte Carlo
0.01	2	100	10000	215
	4	10000	10000	989
	6	10^6	10000	4552
	10	10^{10}	10000	96518
0.0001	2	10000	10^8	30349
	4	10^8	10^8	279520
	6	10^{12}	10^8	$2.57448\ 10^6$
	10	10^{20}	10^8	$2.18394\ 10^8$
10^{-6}	2	10^6	10^{12}	$3.71692\ 10^6$
	4	10^{12}	10^{12}	$5.13512\ 10^7$
	6	10^{18}	10^{12}	$7.09443\ 10^8$
	10	10^{30}	10^{12}	$1.3541\ 10^{11}$

Table 1.1: Number of sampling points required to achieve some error bounds.

- The study of methods for constructing and updating the probability density function of uncertain parameters in the model.
- The development and the implementation of methods for propagating the uncertainty through a given model.
- The study and the application of indirect representations of random variables for dealing with stochastic inequality models and constraints.
- The development and the implementation of a method for treating uncertainty in a black-box type model.

Several methods for characterizing and updating the probability density function of uncertain parameters in the model are available. They include the Bayesian approach and the minimum cross-entropy method. Both these methods share the same form of the posterior distribution from a given prior distribution when the content of information is measured in a suitable metric [199]. Clearly, the comparison study of such methods suggests that the choice between those methods should depend on the complexity of their computations.

Central to the development of the proposed method in this research, which is termed the deterministic equivalent modeling method, for propagating the uncertainty through a given model is the use of both the variational approach and the representations of random variables. Furthermore, fundamental to the implementation of that proposed method is the application of a robust and dedicated symbolic manipulation program and a reliable compiler technology. Together, they can reduce the complexity of the construction of a stochastic model as well as its transformation process into the corresponding deterministic equivalent model through the implementation of an appropriate language. The tangible product of this research work is, therefore, a prototype language for the deterministic equivalent modeling method.

In many practical problems, one not only has to deal with equality type models, but also inequality models. Consequently, one of the objectives of this research work is to study, and to assess the applications of indirect representation methods of random variables to those inequality models. As an example, the Edgeworth series expansion [28] and the optimal linear combination of distributions approach [58] are useful for representing the probability density function of slack variables in the inequality model, hence, they can become a part of the deterministic equivalence of such an inequality model.

Another class of models that should be considered is the black-box type models. This class of models constitutes numerous existing programs or subroutines that are written in different kinds of programming languages or environments. Consequently, it is of interest to develop, and to implement a methodology for dealing with that class of models. In the setting of the deterministic equivalent modeling method, one can apply the collocation method [26] to generate the implicit type of the deterministic equivalent model for a black-box type model in consideration. In this case, the collocation method becomes a strong competitor to the Monte Carlo methods because the collocation method requires much fewer sampling points but can retain the accuracy of the result. As a result, two to three orders of magnitude reduction in computational time can be found in most cases.

1.4 Expected Gain for Including Uncertainty

Eliminating the barriers in any activities involving uncertainty analysis may simplify the application of uncertainty analysis to chemical and environmental engineering systems. However, it is still not sufficient for promoting the use of uncertainty analysis. For practical reasons, people will ask for the real benefit of performing uncertainty analysis. Therefore, in this section, we will see how uncertainty analysis can give us an insight for making a sound decision to achieve a typical objective of chemical and environmental engineering systems.

A typical objective of chemical and environmental engineering systems may involve discontinuous functions. As an example, the objective function of a simple reactor-sizing decision problem [125] can be represented by a bilinear cost function,

$$\text{Cost} = \begin{cases} C_c D, & DA_y \geq Q_y; \quad \text{success on first attempt} \\ C_c \frac{Q_y}{A_y} + C_p, & DA_y < Q_y; \quad \text{redesign required,} \end{cases}$$

where C_c is the catalyst cost per pound, D denotes the number of pounds of catalyst, A_y represents uncertain catalyst activity, and Q_y is the required number of pounds per day of product. If the reactor is not large enough to produce Q_y, and must be rebuilt so that more catalyst can be added, we have to include a penalty cost C_p which is incurred due to business interruption.

If we decide to ignore uncertainty in the catalyst activity A_y, we will use some "best estimate" for its value. It is not clear whether, in practice, this "best estimate" will be the mean, mode, median, or some other central point of the subjective probability density function of A_y [112]. In this case, let us use the mean value, which means that the number of pounds of catalyst, $D^\star$, is $\frac{Q_y}{E(A_y)}$.

Since the value of catalyst activity is uncertain at the design time, one of the possible metrics for evaluating the cost is the expected cost. In this case, let us assume that we only know the possible range of catalyst activity, $A_{yL} \leq A_y \leq A_{yH}$. From such an information, the maximum entropy method suggests that the least-biased probability density function

for A_y is a uniform distribution. Consequently, the expected cost becomes

$$\begin{aligned} E(\text{Cost}) &= \int_{A_{yL}}^{\frac{Q_y}{D}} \frac{1}{A_{yH}-A_{yL}} \left(C_c \frac{Q_y}{A_y} + C_p \right) dA_y + \\ &\quad \int_{\frac{Q_y}{D}}^{A_{yH}} \frac{1}{A_{yH}-A_{yL}} C_c \, D \, dA_y \\ &= \frac{1}{A_{yH}-A_{yL}} \left(C_c Q_y \ln \left(\frac{Q_y}{A_{yL} D} \right) + C_p \left(\frac{Q_y}{D} - A_{yH} \right) + \right. \\ &\quad \left. C_c D \left(A_{yH} - \frac{Q_y}{D} \right) \right), \end{aligned} \tag{1.1}$$

and in the case of ignoring uncertainty we obtain

$$\begin{aligned} E(\text{Cost})|_{D^\star} &= \frac{1}{A_{yH}-A_{yL}} \left(C_c Q_y \ln \left(\frac{Q_y}{A_{yL} D^\star} \right) + C_p \left(\frac{Q_y}{D^\star} - A_{yH} \right) + \right. \\ &\quad \left. C_c D^\star \left(A_{yH} - \frac{Q_y}{D^\star} \right) \right), \end{aligned} \tag{1.2}$$

where $D^\star = \frac{Q_y}{E(A_y)} = \frac{2Q_y}{A_{yL}+A_{yH}}$.

If we decide to include uncertainty in the catalyst activity A_y in our analysis, the optimal number of pounds of catalyst, $\hat{D}$, should be derived from Bayes' decision,

$$\min_{D} \quad E(\text{Cost}),$$

and in this case we obtain

$$\hat{D} = \frac{Q_y}{2A_{yH}} \left(1 + \sqrt{1 + \frac{4C_p A_{yH}}{C_c Q_y}} \right).$$

Similar to the case of ignoring uncertainty, the optimal expected cost by including uncertainty can be calculated from the following formula:

$$\begin{aligned} E(\text{Cost})|_{\hat{D}} &= \frac{1}{A_{yH}-A_{yL}} \left(C_c Q_y \ln \left(\frac{Q_y}{A_{yL} \hat{D}} \right) + C_p \left(\frac{Q_y}{\hat{D}} - A_{yH} \right) + \right. \\ &\quad \left. C_c \hat{D} \left(A_{yH} - \frac{Q_y}{\hat{D}} \right) \right). \end{aligned} \tag{1.3}$$

The difference between the value of $E(\text{Cost})|_{D^\star}$ and the value of $E(\text{Cost})|_{\hat{D}}$ clearly represents the expected gain from including uncertainty,

$$\text{EVIU} = E(\text{Cost})|_{D^\star} - E(\text{Cost})|_{\hat{D}},$$

where EVIU represents the expected value of including uncertainty [112]. It is our expectation of the difference in loss or gain between an optimal decision ignoring uncertainty, and an optimal decision by including uncertainty. From its definition, the value of EVIU should always be greater than or equal to zero. Therefore,

The explicit inclusion of uncertainty in your analysis should at least not make

you worse off in terms of expected value (neglecting the extra effort involved in the analysis) [112].

The extra effort in performing uncertainty analysis, in fact, can be reduced through the elimination of its barriers, and such a reduction is the aim of this research work.

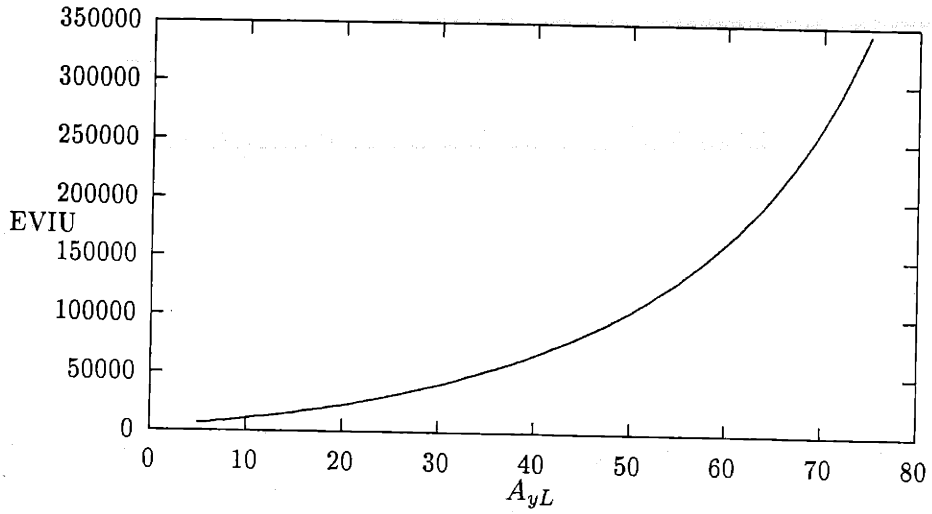


Figure 1-3: Expected Value for Including Uncertainty of reactor-sizing problem, in the case of $C_p = 3 \times 10^5$; $C_c = 10^4$; $Q_y = 10^3$; and $A_{yH} = 100$.

Figure 1-3 shows the value of EVIU as a function of the degree of uncertainty. As the uncertainty increases, the expected gain from including uncertainty in the analysis is decreasing. It means that in the case of high uncertainty, the result from ignoring uncertainty is as good as the result from including uncertainty because in this case the value of penalty cost is relatively higher than the cost of overdesign. Consequently, if we decrease the value of penalty cost in the objective function, the trend can actually change. Figure 1-4 shows the results.

In the case that the value of the penalty cost is relatively smaller than the cost of overdesign, the result from ignoring uncertainty will be as good as of including uncertainty when the uncertainty is low. On the other hand, if the uncertainty increases, the optimal number of pounds of catalyst from ignoring uncertainty is also increasing, and it is always larger than the optimal number of pounds of catalyst from including uncertainty. Since in this case it is better to have an underdesigned reactor, the expected gain for including uncertainty becomes larger in the high uncertainty region. Between these two extreme cases, we also see the case when the magnitude of penalty cost is comparable to the cost of overdesign. In such a case, the optimal number of pounds of catalyst from including uncertainty is located inside the possible range of the optimal number of pounds of catalyst from ignoring uncertainty. Therefore, when those two optimal values coincide, the value of EVIU becomes zero.

The above simple example illustrates the usefulness of uncertainty analysis. In most cases, the value of EVIU is not equal to zero. This also means that in the presence of

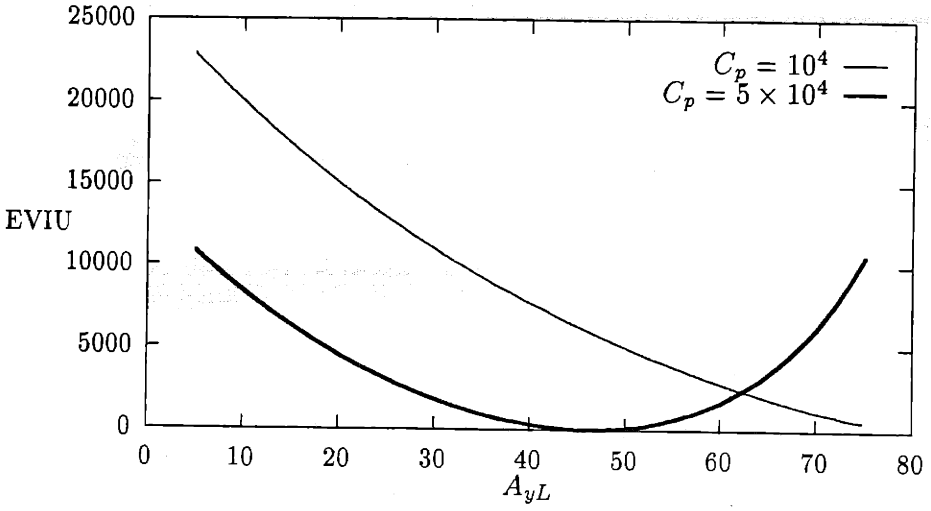


Figure 1-4: Expected Value for Including Uncertainty of reactor-sizing problem, in the case of $C_p = 10^4$ or $C_p = 5 \times 10^4$; $C_c = 10^4$; $Q_y = 10^3$; and $A_{yH} = 100$.

uncertainties, we always have to consider and include such uncertainties in our analysis, otherwise potentially useful information may be lost.

1.5 Thesis Outline

The organization of this thesis in the context of uncertainty analysis is illustrated in figure 1-5. This figure is useful for making a link between each chapter with the problems in uncertainty analysis. It also provides the connection between chapters. To complement that figure, table 1.3 describes the main topics of each chapter.

Critical contents of this research work are given in chapters 3 to 11. While chapters 4 to 7 discuss theoretical components of the deterministic equivalent modeling method, chapter 8 describes current implementation of the method. Furthermore, in chapter 9, typical models in chemical and environmental engineering systems are solved by the deterministic equivalent modeling method. Table 1.2 organizes those examples according to their structures. The last chapter highlights the conclusions from this research work and describes some direction for future research.

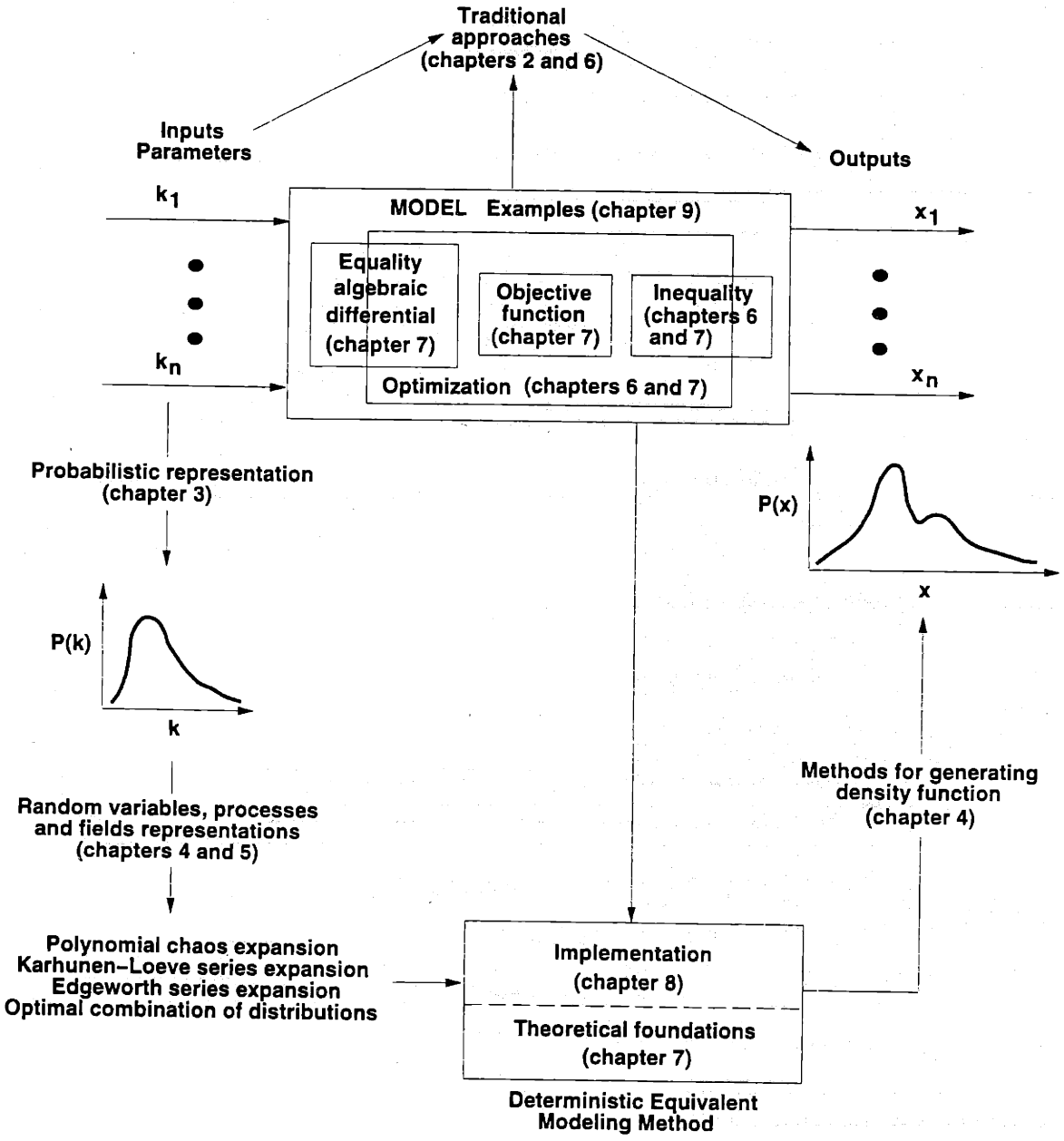


Figure 1-5: Outline of thesis in the context of problems in uncertainty analysis.

Model	Example
Equality, algebraic	Example 9.1.1 Simple algebraic equations model
	Example 9.1.2 Climate forcing by anthropogenic aerosols
	Example 9.1.3 Small gas-phase process flowsheet
Equality, differential	Example 9.1.4 Multiple stationary points
	Example 9.1.5 Photochemical smog prediction
	Example 9.1.6 H_2/O_2 reduced mechanism
Inequality	Example 9.2.1 Linear inequality model
	Example 9.2.2 Nonlinear inequality model
Optimization	Example 9.3.1 Reactors sequence problem
	Example 9.3.2 Stochastic transportation problem

Table 1.2: Structures of models.

Chapter	Topics
1.	– Barriers for performing uncertainty analysis – Objectives of research work – Expected gain for including uncertainty measures the motivation for performing uncertainty analysis
2.	– Numerical approximation methods for stochastic equality models – Random numbers generators and variance reduction techniques
3.	– Parametric uncertainty representations which include interval mathematics, Fuzzy theory, Dempster-Shafer method, and probabilistic approach – Bayesian approach and minimum cross-entropy method for characterizing and updating parametric uncertainty
4.	– Indirect and direct representations methods to represent random variables – Least-square method for computing the values of coefficients of uncertain parameters – Methods for generating probability density functions
5.	– Karhunen-Loève series expansion to approximate random processes and fields
6.	– Recourse technique and chance-constrained method formulations of stochastic inequality models – Stochastic inequality constraints in the optimization model
7.	– Deterministic equivalent modeling method for different types of models including black-box type models
8.	– Implementation of the deterministic equivalent modeling method through the use of symbolic manipulation and compiler technology
9.	– Examples for the applications of the deterministic equivalent modeling method to various chemical and environmental engineering systems
10.	– Recommendation for future related research work
11.	– Conclusions from this research work

Table 1.3: A brief description of topics in each chapter.

Chapter 2

Traditional Approaches

One key result from performing uncertainty analysis is the construction of the probability density function or other interesting attributes of uncertain response variables based on the information of the probability density functions of the uncertain parameters or inputs to the model. Therefore, a number of techniques have been developed for propagating the uncertainty in the parameters and the inputs to the outputs of a given stochastic model. Since the focus of this research work is the development and the implementation of the deterministic equivalent modeling method for propagating the uncertainty through a given model, a discussion of prior works or existing methods for such a problem is apparently critical.

2.1 Numerical approximation methods for stochastic models

In this thesis, three classes of typical models or problems in the field of chemical and environmental engineering are considered. These classes are equality models, inequality models, and optimization models. The first class consists of differential and algebraic equations, and existing numerical methods to deal with them include the perturbation method, the moments method, the hierarchy technique, the stochastic Green's function or the Neumann expansion approach, the semi-group operator method, the spectral-based finite element method. A brief discussion of each of these methods is given in the following text.

Perturbation method [81, 92]

This method assumes that stochastic equations are generated from a *small* perturbation of the deterministic equations. Consequently, this method may not be able to produce an accurate result for a largely perturbed system. To illustrate this method, let us consider a simple example of an algebraic equation:

$$a(\omega)\ x(\omega) \quad = \quad b(\omega),$$

where $a(\omega)$ and $b(\omega)$ are independent random variables whose statistical moments are finite. First, the algebraic equation is rewritten as

$$(E(a) + \epsilon\ a_1(\omega))\ x(\omega) \quad = \quad b(\omega),$$

where $\epsilon\ a_1(\omega) = a(\omega) - E(a)$, and $E(\cdot)$ denotes the expected value operation. Next, it is assumed that the response variable $x(\omega)$ can be expressed as a power series in the parameter ϵ,

$$x(\omega) = \sum_{i=0}^{\infty} x_i(\omega)\ \epsilon^i.$$

Substituting this into the algebraic equation, and separating coefficients of like powers of ϵ, we have a sequence of algebraic equations:

$$\begin{aligned} E(a)\ x_0(\omega) &= b(\omega) && (2.1)\\ E(a)\ x_1(\omega) + a_1(\omega)\ x_0(\omega) &= 0 && (2.2)\\ E(a)\ x_2(\omega) + a_1(\omega)\ x_1(\omega) &= 0 && (2.3)\\ &\vdots \end{aligned}$$

Upon taking the expected value of equation (2.1), we can obtain a deterministic equation for solving the value of $E(x_0)$,

$$E(x_0) = \frac{E(b)}{E(a)}.$$

In order to use equation (2.2) in a similar way to calculate $E(x_1)$, we must first find the value of $E(a_1\ x_0)$. To obtain such a value, we multiply equation (2.1) by $a_1(\omega)$, take the expected value of the resulting equation, and solve for $E(a_1\ x_0)$,

$$E(a_1\ x_0) = \frac{E(a_1\ b)}{E(a)}.$$

Similarly, to calculate the value of $E(x_2)$ from equation (2.3), we first have to calculate the value of $E(a_1\ x_1)$ which in turn requires the value of $E(a_1^2\ x_0)$. We may continue in this fashion to calculate the values of $E(x_3)$, $E(x_4)$, ..., however, the amount of computations required increases as more and more terms needed. Therefore, it is usually necessary to truncate the representation of $x(\omega)$.

In this method, the truncated deterministic equivalent equations are solved successively from the top to the bottom. Furthermore, if the values of successive coefficients in a perturbation expansion decrease rapidly enough, a very good approximation to the true expected value may result.

Moments method [48, 112]

In some cases, the values of moments of response variables are sufficient to describe the propagation of uncertainty in the model. Therefore, the use of moments method for such cases seems appropriate. By its definition, a moment of a random variable is closely related to the expected deviation from the mean value of the random variable. It means that we may apply the Taylor series expansion, which is commonly used to approximate deviations of some variable from its nominal value, to the calculation of moments. To review this method, let us visit the same problem

$$a(\omega)\ x(\omega) = b(\omega).$$

First, the algebraic equation is rewritten as

$$x(\omega) = \frac{b(\omega)}{a(\omega)},$$

and the Taylor series expansion can be invoked at this point to approximate the moments of response variable $x(\omega)$. Before we approximate the moments of $x(\omega)$, it would be useful to describe the use of Taylor series expansion for approximating a general function $y = f(z_1, z_2, \ldots, z_n)$ of n variables,

$$y = f(\mathbf{z})|_{\bar{\mathbf{z}}} + \sum_{i=1}^{n}(z_i - \bar{z}_i)\,\frac{\partial y}{\partial z_i}|_{\bar{\mathbf{z}}} + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(z_i - \bar{z}_i)\,(z_j - \bar{z}_j)\,\frac{\partial^2 y}{\partial z_i\,\partial z_j}|_{\bar{\mathbf{z}}} + \cdots,$$

where $\bar{z}_i$ denotes the nominal value of z_i. In the case of random variable $x(\omega)$, its moments can be approximated by a truncated Taylor series expansion. As an example, two-term Taylor series expansion produces the following approximation,

$$\begin{aligned} E(x) &= \frac{E(b)}{E(a)} + \sigma(a)\,\frac{E(b)}{E^3(a)} - \sigma(a\,b)\,\frac{1}{E^2(a)} \\ E(x^2) &= \frac{E^2(b)}{E^2(a)} + \sigma(a)\,\frac{3\,E^2(b)}{E^4(a)} + \sigma(b)\,\frac{1}{E^2(a)} - \sigma(a\,b)\,\frac{4\,E(b)}{E^3(a)} \\ &\vdots \end{aligned}$$

where $\sigma(\cdot)$ represents the covariance or the variance function.

Similar to the perturbation method, the application of the moments method is limited by the shortcoming of the Taylor series expansion. If the variances of uncertain parameters $a(\omega)$ and $b(\omega)$ are relatively large, the truncated Taylor series expansion may not be sufficiently accurate.

Hierarchy technique [92]

The hierarchy technique is another approximation method that also generates a set of deterministic equivalent equations. This set of equations is obtained by correlating the response variable with the uncertain parameters. To gain some understanding of this method, we shall apply a simple hierarchy technique to the same problem,

$$a(\omega)\,x(\omega) = b(\omega).$$

First, the algebraic equation is rewritten as

$$x(\omega) + \hat{a}(\omega)\,x(\omega) = b(\omega),$$

where $\hat{a}(\omega) = a(\omega) - 1$. Next, a hierarchy of equations may be obtained as follows:

$$x(\omega) + \hat{a}(\omega)\,x(\omega) = b(\omega) \tag{2.4}$$

$$\hat{a}(\omega)\,x(\omega) + \hat{a}^2(\omega)\,x(\omega) = \hat{a}(\omega)\,b(\omega) \tag{2.5}$$

$$\hat{a}^2(\omega)\,x(\omega) + \hat{a}^3(\omega)\,x(\omega) = \hat{a}^2(\omega)\,b(\omega) \tag{2.6}$$

$$\vdots$$

After taking the expected value of equation (2.4), we can calculate the value of $E(x)$ if $E(\hat{a}\ x)$ is known. Now, the value of $E(\hat{a}\ x)$ can be obtained by taking the expected value of equation (2.5). However, in order to solve equation (2.5), we need the value of $E(\hat{a}^2\ x)$ which should come from equation (2.6). It is clear at this point that the solution to each equation in the hierarchy depends on the solution of the next equation in the hierarchy. Therefore, some assumption must be made to allow this set of equations to be solved by itself. As an example, we may choose to assume that

$$E(\hat{a}^3\ x) \quad = \quad E(\hat{a}^3)\ E(x).$$

Such an assumption is called a correlation discard. Clearly, other assumptions can be made which also allow the truncation of a hierarchy. This simple example reveals the weakness of the hierarchy technique. The correlation discard assumption in general is incorrect, and it only can be applied only to cases with small randomness [92].

Neumann expansion method [1, 56, 164]

Limitations to the perturbation and hierarchy methods necessitate more general methods, such as the stochastic Green's function or the Neumann expansion method, to be developed. The Neumann expansion method assumes that an operator can be decomposed as a summation of the deterministic and the stochatic operators. Furthermore, the inverse of such an operator is represented by its Neumann expansion. This method is again applied to the same simple problem,

$$a(\omega)\ x(\omega) \quad = \quad b(\omega).$$

First, this algebraic equation is rewritten as

$$(E(a) + \dot{a}(\omega))\ x(\omega) \quad = \quad b(\omega),$$

where $\dot{a}(\omega) = a(\omega) - E(a)$. According to the method, the deterministic operator in this problem is $E(a)$, and the stochastic operator is $\dot{a}(\omega)$. At this point, a Neumann expansion of the inverse operator may be performed, leading to

$$\begin{aligned} x(\omega) &= (E(a) + \dot{a}(\omega))^{-1}\ b(\omega) \\ &= \sum_{i=0}^{\infty} \left(E(a)^{-1}\ \dot{a}(\omega)\right)^i\ E(a)^{-1}\ b(\omega). \end{aligned}$$

This result is valid provided that

$$\|E(a)^{-1}\ \dot{a}(\omega)\| \quad < \quad 1, \tag{2.7}$$

where $\|\cdot\|$ represents some norm in the $R \times \Omega$ space. Condition (2.7) is inherited from the Neumann expansion, and it becomes the main limitation of the Neumann expansion method.

Semi-group operator method [142, 143]

To eliminate the condition posed in the Neumann expansion method, Serrano [143] suggested the use of semi-group operator method to solve a stochastic evolution equa-

tion of the form

$$\frac{dx(t,\omega)}{dt} + A(\omega)\, x(t,\omega) \;=\; 0,$$

with initial condition

$$x(0,\omega) \;=\; x_0(\omega).$$

Assume that there exists an operator J_t which assigns to each $x_0(\omega)$, the value of $x(t,\omega)$ at time $t > 0$,

$$x(t,\omega) \;=\; J_t\, x_0(\omega).$$

From the uniqueness property of the solution, we obtain the following property for the operator:

$$J_{t+s} \;=\; J_t\, J_s.$$

An operator with this property is called a semi-group. Furthermore, if J_t is an operator on R^+, and satisfies:

1. $J_{t+s} = J_t\, J_s; \quad t, s \geq 0.$
2. $J_0 = I$, the identity operator.
3. $\|J_t x - x\|_X \to 0$ as $t \to 0$, for all x in a Banach space X.

then J_t is said to be a strongly continuous semi-group. In the case of an inhomogeneous evolution equation,

$$\frac{dx(t,\omega)}{dt} + A(\omega)\, x(t,\omega) \;=\; b(t,\omega),$$

the solution can be written as

$$x(t,\omega) \;=\; J_t\, x_0(\omega) \;+\; \int_0^t J_{t-s}\, b(s,\omega)\, ds.$$

Since the semi-group operator method searches for the semi-group operator of the stochastic evolution equation in consideration, this method can avoid the restrictions imposed by approximating the inverse of the equation operator. Unfortunately, the derivation of the semi-group operator requires knowledge of functional analysis and other rigorous mathematical treatments. Moreover, different forms of evolution equation create different semi-group operators, and different boundary conditions may also produce different semi-group operators.

Spectral-based finite element method [56]

Similar to the case of semi-group operator method, the spectral-based finite element method has been developed to deal with stochastic differential equations. It is mainly used for assessing the reliability of a structure which is exposed to a random vibration or foundation. A major contribution of this approach is the application of the polynomial chaos expansion and the Karhunen-Loève series expansion to directly represent uncertain parameters. Such representations allow the use of spectral method in determining the response variables.

In solving a typical structure problem which is usually posed as a boundary value problem, the spectral-based finite element method involves two stages of computations. In the first stage, the Karhunen-Loève series expansion provides an optimal mean-square convergent representation of a stochastic process in terms of a denumerable set of uncorrelated random variables. Furthermore, the solution is approximated by applying the finite element method. Consequently, the end product from the first stage is a matrix equation involving deterministic matrices multiplied by random variables,

$$\left[I + \sum_{k=1}^{M} \xi_k \, Q^{(k)}\right] \; x \;\; = \;\; g,$$

where

$$\begin{aligned} Q^{(k)} &= \left[K^{(0)}\right]^{-1} \; K^{(k)} \\ g &= \left[K^{(0)}\right]^{-1} \; f. \end{aligned}$$

and ξ_k is the k-th random variable in the Karhunen-Loève series expansion. The matrix $K^{(k)}$; $k = 0, \ldots, M$ and the vector f are calculated from the finite element formulation.

The second stage in the solution approach addresses the issue in solving such a matrix equation. One option is to use a Galerkin scheme whereby the solution is approximated by its projection onto a complete basis in the space of random variables,

$$x \;\; = \;\; \sum_{i=0}^{P} x_i \Psi_i(\{\xi\}),$$

Such a basis is identified as the polynomial chaos expansion $\Psi_i(\{\xi\})$. In this case, a set of algebraic equations is obtained

$$\sum_{k=1}^{M} \sum_{i=0}^{P} E\left(\xi_k \; \Psi_i(\{\xi\}) \; \Psi_j(\{\xi\})\right) \; Q^{(k)} \; x_i \;\; = \;\; E\left(\Psi_j(\{\xi\})\right) \; g; \quad j = 1, \ldots, P,$$

from which the solution can be computed.

This novel method actually can be reformulated in the setting of deterministic equivalent modeling method. Instead of applying the Galerkin's approach to a matrix equation as a specific approximation of the original model which depends on the chosen numerical method, this thesis proposes a direct projection of the original stochastic model onto the space of polynomial chaos. As a result, users of the deterministic equivalent modeling method can have more alternatives in selecting desirable deterministic numerical methods to solve the deterministic equivalent equations. Another advantage of the deterministic equivalent modeling method is that the projection scheme can be applied in part to other types of models, such as inequality or optimization models.

Another class of methods for solving stochastic differential equations is the class of direct methods. They are called the direct methods because the application of these methods to a stochastic differential equation results in a transition probability density function.

Two widely known methods from this class are the Fokker-Planck, and the Liouville approaches [51]. The earlier method is used to solve the stochastic differential equations whose parameters are uncertain, and the latter one is applied to the case of uncertain initial conditions. A limitation of these methods is that the response variables are assumed to be a Markov process.

Stochastic equality models may also be solved through the use of statistical sampling techniques which are usually known as Monte Carlo methods [38]. The key elements of these methods, such as random number generators and variance reduction techniques, are discussed in the following sections. While the Monte Carlo methods are simple and easy to use, they may be prohibitively expensive for some problems, such as a stochastic optimization problem with deterministic decision variables. Except for cases for which the evaluation of model is inexpensive, should such techniques be used as the last resort. This thesis proposes the use of collocation method in the framework of the deterministic equivalent modeling method as a much cheaper alternative to the Monte Carlo methods for solving a black-box type stochastic model.

A discussion of existing numerical approximation techniques for dealing with stochastic inequality and optimization models can be found in chapter 6. References [45, 160, 176, 177] provide the essence and the background of those techniques.

2.2 Monte Carlo methods

According to Doll and Freeman [39], in the early 1950's, Metropolis *et al.* devised an ingenious stochastic method for the evaluation of probability distribution weighted averages. Their approach was a special case of what have become known as Monte Carlo methods. In these methods, as can be guessed from the colorful name established by Ulam [174], one devises a suitable random "game" in which the answer to the question under study emerges as the game is played. Therefore, Monte Carlo methods, or methods of statistical trials, refer to procedures or methodologies for solving various problems of computational mathematics by means of the construction of some random processes for each of such problems, with the parameters of the process equal to the required quantities of the problem [23].

There are two main classes into which Monte Carlo calculations usually fall. In the first class, the problems to be solved have some physical probabilistic structures and, in essence, the random numbers are used in such a way that they more or less directly simulate the physical situations. The second class of Monte Carlo calculations concerns with problems that at first sight have no probabilistic features. It means that one should first derive a different problem which has the same numerical answer as the original problem and also has a probablistic structure. In other words, one should search the duality of the problems in the probabilistic problems space. It then suffices to solve the problem by simulation or statistical trials, as in the first class of Monte Carlo calculations. A more detailed description and survey of the Monte Carlo methods can be seen in Halton's paper [62].

It is clear that Monte Carlo methods require random number generators for simulating the probabilistic structures in the problems. In the development and the use of such generators, there are two issues that should be considered. The first issue is the optimality of the random numbers. To consider such an issue, we need to use an optimal set of random numbers that yields the most accurate result using the fewest number of random points. Another issue deals with reducing the probablistic error of the results of simulation. This second issue has its own field of study: the variance reduction techniques. The next sec-

tion will discuss random number generators in general. Following that section, techniques to transform uniform to non-uniform random points are then described. The last section will highlight several variance reduction methods. An example of the use of Monte Carlo methods for solving a Bayesian problem can be seen in the next chapter.

2.3 Random number generators

There are two main classes of practical random number generators [69]:

- Pseudo-random numbers are random numbers produced by a simple numerical algorithm such as congruential or shift-register methods. These random numbers are not truly random. Nevertheless, any given sequence of pseudo-random numbers may appear random to someone who does not know the algorithm.
- Quasi-random numbers are designed to be more uniformly distributed than pseudo-random numbers (see figure (2-1)), and to give deterministic error bounds [115]. This property is used to reduce the errors in some applications of Monte Carlo methods. As an example, this type of random numbers has been applied to calculate multi-dimensional integrals.

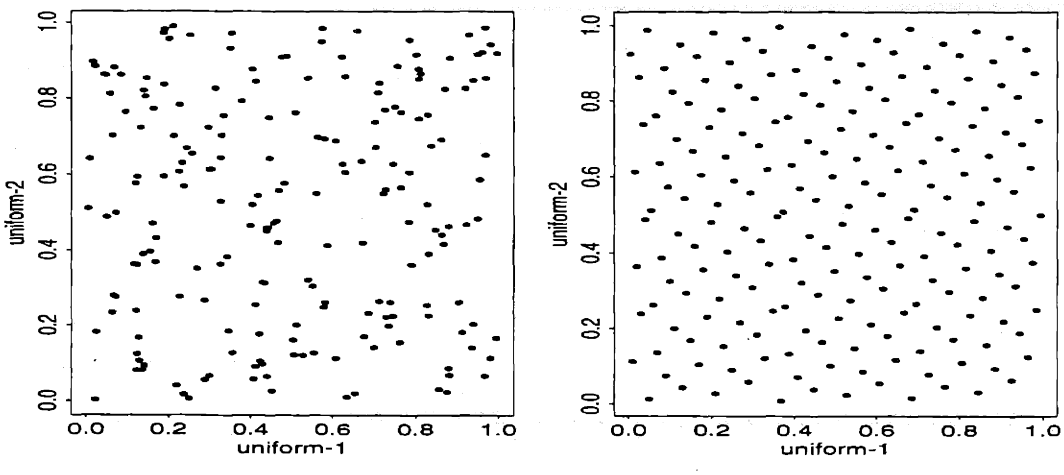


Figure 2-1: Scatter plots for two dimensional 200 uniform random numbers: the left figure is generated by a standard C-language pseudo-random number generator; the right figure is generated by the Hammersley-Wozniakowski sampling method, a quasi-random numbers generator.

There are three classes of simple pseudo-random number generators that have been used most extensively [69, 136]. These are known as: multiplicative linear congruential, Fibonacci, and shift register or Tausworthe generators. An excellent reference for the congruential generators is a book by Knuth [83]. This reference contains theoretical and practical considerations for choosing the right multipliers in the generator, and the testing procedures for pseudo-random number generators as well.

A paper by Marsaglia, Narasimhan and Zaman [102] can be referred to for the description of the Fibonacci generators. A similar generator which has been proposed by Marsaglia,

Zaman, and Tsang combines lagged Fibonacci and "arithmetic" sequences [69]. The result is a portable generator, such that it produces bit-identical results on all machines with at least 24-bit mantissas in the floating-point representation, and the period for this generator is approximately 2^{144}.

An example of shift register generators is **random**, a random number generator that comes from the standard **C** programming language library. It is based on a non-linear additive feedback algorithm which employs a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. This generator has a long period, approximately 16 $(2^{31}-1)$. A more detailed description and an implementation of the shift register pseudo-random number generator can be found in a paper of Lewis and Payne [94].

Wozniakowski recently has devised a procedure to generate an optimal set of sampling points for computing multidimensional integrations [193]. This set of sampling points is derived from quasi-random numbers known as Hammersley points which in turn is a generalization of the Van der Corput's sequence. The mechanism for generating such optimal sampling points will be described below, while the theoretical backgound is referred elsewhere [61, 193].

Let R be any integer, then any other integer n can be written in radix-R notation as

$$n \equiv n_M n_{M-1} \ldots n_2 n_1 n_0 = n_0 + n_1 R + n_2 R^2 + \cdots + n_M R^M,$$

where

$$M = [\log_R(n)] = \left\lceil \frac{\log(n)}{\log(R)} \right\rceil,$$

and the square bracket denotes the integral part. By reversing the order of the digits for n, or in other words, by taking the mirror image of these digits relative to the demical point, one can uniquely construct a fraction between 0 and 1. This reversing procedure is known as the radical inverse function, $\phi_R(\cdot)$.

$$\phi_R(n) = 0.n_0 n_1 n_2 \ldots n_M = n_0 R^{-1} + n_1 R^{-2} + \cdots + n_M R^{-M-1}.$$

Van der Corput's sequence of N points in a unit square can be written as $(\frac{n}{N}, \phi_2(n))$ for $n = 0, 1, \ldots, N-1$. For generating a sequence of N points in a k-dimensional unit hypercube, Hammersley suggested the use of the following formula,

$$z_k(n) = \left(\frac{n}{N}, \phi_{R_1}(n), \phi_{R_2}(n), \ldots, \phi_{R_{k-1}}(n)\right); \quad n = 0, 1, \ldots, N-1,$$

where $R_1, R_2, \cdots, R_{k-1}$ are the first $k-1$ prime numbers.

Wozniakowski proposed a shifted version of Hammersley points to be the optimal sampling points in the average case complexity measure. These optimal N sampling points can be written as follows:

$$x_k(n) = \left(1 - \frac{n+t}{N}, 1 - \phi_{R_1}(n), 1 - \phi_{R_2}(n), \ldots, 1 - \phi_{R_{k-1}}(n)\right); \quad 0 \leq n+t < N,$$

where t is a constant. We can assume $t = 0$, and if we want to have random numbers inside the unit hypercube, we may use the following formula,

$$\acute{x}_k(n) = \left(1 - \frac{n}{N+1}, 1 - \phi_{R_1}(n), 1 - \phi_{R_2}(n), \ldots, 1 - \phi_{R_{k-1}}(n)\right); \quad n = 1, 2, \ldots, N.$$

It is clear that if $N \gg 1$ and $t = 0$, the maximum difference between $x_k(n)$ with $\acute{x}_k(n)$ will be $O\left(N^{-1}\right)$. The optimality of a set of sampling points can be measured in terms of the number of points needed as the error of computing multi-dimensional integrations, ε, goes to zero. Wozniakowski has found that as $\varepsilon \to 0$, the number of point $x_k(n)$ required is

$$N_x = O\left(\frac{1}{\varepsilon} |\log(\varepsilon)|^{\frac{k-1}{2}}\right),$$

and similarly for the case of $\acute{x}_k(n)$, we have

$$N_{\acute{x}} = O\left(\frac{1}{\varepsilon} |\log(\varepsilon)|^{k-1}\right).$$

It is of interest to note that the number of points required to obtain the same accuracy using standard Monte-Carlo techniques, or using a uniform grid are [200]

$$N_{MC} = O\left(\frac{1}{\varepsilon^2}\right)$$
$$N_{UG} = O\left(\frac{1}{\varepsilon^k}\right).$$

In practical applications, it may be more convenient to delete the first coordinate of the points since we do not have to recalculate the first component of each point if we decide to increase N, and it is easier to monitor the error of approximation during the computation process. The formula then becomes

$$\grave{x}_k(n) = (1 - \phi_{R_1}(n), 1 - \phi_{R_2}(n), \ldots, 1 - \phi_{R_k}(n)); \quad n = 1, 2, \ldots, N.$$

This formula can be recognized as the procedure for generating the shifted Halton points, and its optimality is given as follows:

$$N_{\grave{x}} = O\left(\frac{1}{\varepsilon} |\log(\varepsilon)|^{k}\right).$$

Several implementations for this Hammersley-Wozniakowski method can be seen elsewhere [164, 76]. Figure 2-1 shows an example of two dimensional Hammersley-Wozniakowki sampling points. These points are more uniform than the random points from the standard **C** programming language subroutine. Nonetheless, they are less independent than those of the standard **C** programming language. We can improve the independence property of successive Hammersley-Wozniakowski points by using a "shuffling" technique [76]. Since our purpose is to compute the distribution of dependent variables in the model, we may use the "unshuffled" version.

There are many statistical tests to investigate statistical properties of a random number generator. As an example, we perform Kolmogorov-Smirnov, χ^2, permutation, and maximum tests to both the standard **C** programming language random numbers generator and

the Marsaglia, Zaman, and Tsang (MZT) random numbers generator. The following table shows a summary for those tests.

Test	Standard C generator	MZT generator
KS test (K^+,K^-)	(0.233356, 0.59148)	(0.525605, 0.681511)
χ^2 test with 100 bins	91.4	120.2
Permutation 2 test	1.156	0.4
Permutation 4 test	24.736	18.88
Maximum 2 test (K^+,K^-)	(0.353321, 0.458979)	(0.397775, 0.922231)

Table 2.1: Empirical tests for 2000 pseudo-random numbers; seed number for standard C generator is 1, and for MZT is (1,2).

Based on the limiting values given by Knuth [83] for the corresponding test, we conclude that both generators perform quite well on those tests. However, since the period for MZT generator is much longer than that of the standard C programming language generator, the MZT generator is more suitable for applications that need long period random numbers.

Similarly, statistical tests can be performed to the quasi-random numbers. As an example, Kalagnanam and Diwekar have conducted the serial and runs-down tests to Hammersley-Wozniakowski sampling points [76]. The results from these tests show that the Hammersley-Wozniakowski sampling points are more uniform, but less independent between successive points than linear congruential random numbers. However, they also showed that the "shuffled" Hammersley-Wozniakowski sampling points may have good independence property.

2.4 Non-uniform variate

In the previous section, we have discussed several common random number generators. These generators usually produce uniform distributed random numbers between zero and one. Many practical applications however require random numbers with non-uniform distribution. Therefore, some general or specialized techniques can be used to transform uniform random numbers to non-uniform random numbers [37]. There are three general techniques that regularly used for generating non-uniform random numbers. They include: inverse transform method, composition method, and acceptance-rejection method [138]. These three techniques will be described below along with several specialized techniques for generating univariate normal, log-normal, and exponential distributed random numbers. Many alternative and different techniques for generating random numbers from other types of distributions, and for multivariate cases can be found elsewhere [34, 37, 138]

2.4.1 Inverse transform method

The inverse transform method is based upon the following theorem [37]:

Theorem 2.1 *Let $F(\cdot)$ be a continuous cumulative distribution function on R with inverse $F^{-1}(\cdot)$ defined by*

$$F^{-1}(u) = \inf\{x|F(x) = u, 0 < u < 1\}.$$

If U is a uniform $[0,1]$ random variable, then $F^{-1}(U)$ has distribution function $F(\cdot)$. Also if X has distribution function $F(\cdot)$, then $F(X)$ is uniformly distributed on $[0,1]$.

The above theorem provides a mechanism for generating random numbers with cumulative distribution function $F(\cdot)$:

1. Generate a uniform $[0,1]$ random number u.
2. Return $x \leftarrow F^{-1}(u)$.

This method clearly requires the knowledge of $F^{-1}(\cdot)$. For some distribution functions, we can explicitly obtain the form of $F^{-1}(\cdot)$, and for others, we need to count on the inversion by numerical solution of $F(x) = u$, or explicit approximations of the inverse function [37]. Several areas where the inversion method finds its niche are the generation of correlated random numbers, and the generation of maxima or order statistics. It should be pointed out that even in the case when $F^{-1}(\cdot)$ exists in an explicit form, the inverse transform method is not necessarily the most efficient method for generating non-uniform random numbers [138]. As an example of the inverse transform method, we will consider the problem of generating random numbers from the piece-wise constant probability density function [138].

Example 2.4.1.1

A random variable, x, has the following probability density function:

$$f_x(x) = \begin{cases} c_i, & x_{i-1} \le x \le x_i;\ i = 1,2,\ldots,n \\ 0, & \text{otherwise} \end{cases}$$

where $c_i \ge 0$, and $x_0 < x_1 < \cdots < x_{n-1} < x_n$. By definition, the cumulative distribution function of x can be written as follows,

$$F_x(x) = \sum_{j=1}^{i-1} \int_{x_{j-1}}^{x_j} f_x(x)\, dx + \int_{x_{i-1}}^{x} c_i\, dx,$$

where $i = \max_j \{j | x_{j-1} \le x\}$. The inverse transform method requires the solution of $F_x(x) = u$ with respect to x,

$$x = x_{i-1} + \frac{u - \sum_{j=1}^{i-1} P_j}{c_i},$$

where $P_j = \int_{x_{j-1}}^{x_j} f_x(x)\, dx$, and the value of u satisfies the following constraint,

$$\sum_{j=1}^{i-1} P_j \le u \le \sum_{j=1}^{i} P_j.$$

The implementation of this result becomes:

1. Generate a uniform $[0,1]$ random number u.
2. Find i from

$$\sum_{j=1}^{i-1} P_j \le u \le \sum_{j=1}^{i} P_j, \quad i = 1,\ldots,n.$$

3. Return $x \leftarrow x_{i-1} + \frac{u - \sum_{j=1}^{i-1} P_j}{c_i}$.

2.4.2 Composition method

The main purpose of the composition or decomposition method is to generate random numbers for a target density $f(\cdot)$ from a discrete or continuous mixture of different random numbers with much simpler density function $f_i(\cdot)$. A continuous version of this method can be written as follows [138]:

$$f_x(x) = \int g(x|y)\, dF_y(y),$$

where $f_x(\cdot)$ is the target density, and y is a random number with simpler cumulative distribution function $F_y(\cdot)$. The above equation means that if a value of y is drawn from a continuous cumulative distribution function $F_y(y)$, and then if x is sampled from the $g(x)$ for that chosen y, the density function for x will be $f_x(x)$. Similarly, the discrete version for this method has the following form

$$f_x(x) = \sum_{i=1}^{\infty} f_i(x)\, p_i,$$

where $f_i(x) = g(x|y = i)$, and p_i is the probability of $y = i$.

We may apply the continuous version of the composition method to generate random numbers whose distribution has integral form. Similarly, the discrete version is usually used to decompose a complicated distributed random number into several random numbers with simpler distributions. A combination of the composition and inverse transform methods is useful for generating random numbers from a complicated form of density function. The composition method first decomposes the support of the density function into several intervals where the inverse of local density function at each interval can be calculated effortlessly. We then apply the inverse transform method to generate random numbers from those intervals, and choose from the generated random numbers according to the probability of intervals. Different examples for the application of this method can be seen in Rubinstein's book [138] and others [34, 37].

2.4.3 Acceptance-rejection method

This method was introduced by Von Neumann [185], and based on the following theorem [138].

Theorem 2.2 *Let x be a random variate distributed with the probability density function $f_x(x)$ which is represented as*

$$f_x(x) = C\, g(x)\, h(x),$$

where $C \geq 1$, $0 < g(x) \leq 1$, and $h(x)$ is also a probability density function. Now, let u and y be distributed uniform $[0, 1]$ and $h(y)$ respectively. Then

$$f_y(x|u \leq g(y)) = f_x(x).$$

A mechanism that follows the above theorem can be develop to implement the acceptance-rejection method:

1. Generate u from uniform $[0, 1]$.
2. Generate y from the probability density function $h(y)$.
3. If $u \leq g(y)$, return $x \leftarrow y$, otherwise go back to step 1.

The efficiency of this method is determined by the third step of the mechanism. Increasing the probability of $u \leq g(y)$, or decreasing the value of C, will raise the efficiency of the acceptance-rejection method. Clearly, if the efficiency becomes 100% or $C = 1$, we can eliminate the third step since $h(x) = f_x(x)$. Another important point to be considered in using this method is the difficulty to generate random numbers from the density function $h(x)$. In other words, there is a need to assess the trade-off between the ease of generating random numbers from $h(x)$ with the efficiency imposed by $g(x)$ before one implements specific forms of $h(x)$ and $g(x)$.

In practice, there are a number of acceptance-rejection methods for generating random numbers from continuous distributions. They include envelope rejection, band rejection, ratio of uniforms, and Forsythe's methods [34]. Similar to the case of composition and inversion methods, we can also have a combination of the composition method and the acceptance-rejection method [37].

2.4.4 Normal variate

A random variable x has a normal distribution if its probability density function is

$$f_x(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right); \quad -\infty < x < \infty,$$

where μ and σ are the mean value and standard deviation of x respectively. In practice, we can generate a specific normal distributed random number, x, with mean value μ and standard deviation σ from a standard normal distributed random number, y, which has mean value and standard deviation equal to 0 and 1 respectively,

$$x = \mu + \sigma y.$$

Therefore, every practical normal variate generator only produces standard normal distributed random numbers.

There is a simple, but inadvisable, procedure for generating standard normal deviates which uses the notion of the central limit theorem. This procedure requires 12 independent uniform variates, $\{u_i; i = 1, \ldots, 12\}$, to generate a standard normal random number y,

$$y = \sum_{i=1}^{12} u_i - 6.$$

This procedure fits poorly in the tails of the distribution, therefore, it should not be used for simulating rare events [34]. Furthermore, the requirement of 12 uniform variates for each normal variate is likely to make such a procedure slow, even when the uniform random numbers generator is fast.

Another simple normal variate generator is the Box-Müller method [19]. Their procedure requires a pair of uniform variates to generate a pair of standard normal variates,

$$y_1 = \cos(2\pi u_2)\ \sqrt{-2\ \ln(u_1)} \tag{2.8}$$

$$y_2 = \sin(2\pi u_2)\ \sqrt{-2\ \ln(u_1)}. \tag{2.9}$$

These equations can be derived by considering a transformation $(y_1, y_2) \rightarrow (r, \theta)$ as follows:

$$y_1 = r\ \cos(\theta), \tag{2.10}$$

$$y_2 = r\ \sin(\theta). \tag{2.11}$$

The joint probability density function for r and θ becomes (see chapter 4)

$$\begin{aligned} f_{r,\theta}(r,\theta) &= \frac{f_{y_1,y_2}(y_1,y_2)}{|J(y_1,y_2)|} \\ &= \frac{1}{2\pi}\ \exp\left(-\frac{1}{2}\left(y_1^2+y_2^2\right)\right)\ |J(y_1,y_2)|^{-1} \\ &= \frac{1}{2\pi}\ \exp\left(-\frac{1}{2}r^2\right)\ r, \end{aligned}$$

where J is the Jacobian of transformation. If we further make a transformation $r \rightarrow s$ as follows:

$$s = \frac{r^2}{2},$$

then the joint probability density function for s and θ has the following form

$$f_{s,\theta}(s,\theta) = \frac{1}{2\pi}\ \exp(-s),$$

which shows that s and θ are independently distributed. In this case, s is a standard exponential variate (see subsection 2.4.6), and θ is a uniform $[0, 2\pi]$ variate. Applying the inversion method, these two random variates may be generated by

$$\begin{aligned} \theta &= 2\pi u_1 \\ s &= -\ln(u_2), \end{aligned} \tag{2.12}$$

and the corresponding r can be generated by

$$r = \sqrt{2s} = \sqrt{-2\ \ln(u_2)}. \tag{2.13}$$

Clearly, substituting equations (2.12) and (2.13) into equations (2.10) and (2.11) yields equations (2.8) and (2.9).

To avoid the use of trigonometric functions that may sometimes slow down this procedure, Marsaglia and Bray [103] modified the procedure as follows:

$$\begin{aligned} y_1 &= \sqrt{-2\ln\left(u_1^2+u_2^2\right)}\ u_1\ \left(u_1^2+u_2^2\right)^{-\frac{1}{2}} \\ y_2 &= \sqrt{-2\ln\left(u_1^2+u_2^2\right)}\ u_2\ \left(u_1^2+u_2^2\right)^{-\frac{1}{2}}. \end{aligned}$$

This modified procedure is known as the polar Box-Müller method, and is only valid when $u_1^2 + u_2^2 \leq 1$, where u_1 and u_2 are uniform $[-1, 1]$. It means that, in average, $\frac{4}{\pi}$ uniform variates are required per normal variate generated [34]. On the other hand, the original Box-Müller method only needs 1 uniform variate per normal variate generated. Therefore, there is a trade-off between increasing the number of uniform variates required and reducing the computational time for the trigonometric functions.

There are other methods for generating normal random numbers [37]. They include: rejection methods by Von Neumann, and Sibuya; Ratio-of-uniforms method by Kinderman and Monahan; Composition-rejection methods by Ahrens and Dieter, Kinderman and Ramage, and Sakasegawa; Series methods; Almost-exact inversion method by Wallace; Table methods by Marsaglia, Maclaren and Bray; Forsythe's method by Forsythe, Ahrens and Dieter, and Brent; Butcher's method [34]. These methods differ in the number of uniform random numbers required, the ease of implementation, and the number of arithmetic operations needed. An appropriate measure of efficiency for these generators is therefore the average computational time or cost required for generating a normal random number. As an example, Dagpunar [34] noted that for **Fortran** implementations, computational evidence suggests that the best of the easily implemented methods are not substantially slower than the best of the more complex methods, on any given machine.

2.4.5 Log-normal variate

Log-normal distributed random numbers are sometimes used to simulate uncertain variables that can take only a half of real line values, (θ, ∞). A log-normal random variable, z, has a density function with support on (θ, ∞) given by

$$f_z(z) = \frac{1}{(z-\theta)\,\sigma\,\sqrt{2\pi}} \exp\left(-\frac{(\ln(z-\theta)-\mu)^2}{2\sigma^2}\right); \quad z > \theta.$$

Since a log-normal random variable can be obtained from a standard normal random variable with the following transformation:

$$z = \theta + \exp(\mu + \sigma y),$$

the mechanism for generating the log-normal variate becomes

1. Generate y from normal distribution with mean zero and unit variance.
2. Return $z \leftarrow \theta + \exp(\mu + \sigma y)$.

The efficiency of this mechanism clearly depends on the efficiency of the standard normal variate generator used in the first step.

2.4.6 Exponential variate

A random variable has a negative exponential distribution if its probability density function is given by

$$f_x(x) = \lambda \exp(-\lambda x); \quad x \geq 0,$$

where $\lambda > 0$. The simplest mechanism for generating an exponential variate is the inversion method,

$$u = F_x(x) = 1 - \exp(-\lambda x),$$

which gives

$$x = -\frac{1}{\lambda} \ln(1-u),$$

and since $1-u$ is distributed in the same way as u, that is uniform $[0,1]$, we have

$$x = -\frac{1}{\lambda} \ln(u).$$

In practice, exponential variate generators produce standard exponential random numbers, y, whose λ equals to 1. One can then transform y into x as follows:

$$x = \frac{1}{\lambda} y.$$

Similar to the case of normal variate generators, there are many alternative exponential generators which eliminate or reduce the need of logarithmic function [34, 37, 138]. One example for such generators is Von-Neumann's ingenious method [185] which was extended by Forsythe. This method is to generate sequences of uniform $[0,1]$ random numbers $\left\{u_1^{(i)}, u_2^{(i)}, u_3^{(i)}, \ldots\right\}$, $(i = 1, 2, \ldots)$, stopping at $i = t$, when $N^{(i)}$ is odd-valued, where

$$N^{(i)} = 1 \quad \text{if } u_1^{(i)} < u_2^{(i)},$$

and

$$N^{(i)} = n \quad \text{if } u_1^{(i)} \geq u_2^{(i)} \geq \cdots \geq u_{n-1}^{(i)} < u_n^{(i)}, \quad n > 2.$$

The standard exponential random number is then

$$x = t - 1 + u_1^{(t)}.$$

By eliminating the need of logarithmic function, this procedure increases the number of uniform random numbers required for generating an exponential random number. The average number of uniform random numbers required is 6 [185]. Dagpunar conducted a comparison study on different platforms between the inversion method and the Von-Neumann-Forsythe method [34]. The variations in the result of this study suggest that it is probably worthwhile to determine which of these two methods is faster in the user's particular environment.

2.5 Variance reduction methods

As mentioned previously, one important issue in the use of Monte Carlo method is how one can reduce the probabilistic error associated with the result. To address this issue, a collection of techniques called variance reduction method has been developed. The aim of these techniques is to reduce the variance of estimation or solution of the problem while

keeping the same number of random numbers generated and used. This aim may be achieved in several ways: by introducing correlation among observations; by embedding the smaller complicated model into a larger simpler model; and by using weighted sampling based on prior qualitative and quantitative information. Variance reduction methods are definitely needed when evaluation of the simulation model is costly, and the need for accurate results is crucial.

2.5.1 Common random numbers technique

Common random number technique reduces the variance in the difference of two output measures by making trials as similar as possible between the two measures [108]. This technique is important for reducing the variance of estimating the difference between measures from different models or systems. As an example, if x_1 and x_2 are measures from models 1 and 2 respectively, then the variance of the estimator of their difference becomes

$$\sigma^2(\theta) \quad \propto \quad \sigma^2(x_1) + \sigma^2(x_2) - 2\,\sigma^2(x_1, x_2),$$

where $\sigma^2(\cdot)$ and $\sigma^2(\cdot,\cdot)$ are variance and covariance functions respectively, and θ is the estimator of $x_1 - x_2$. It is clear that if we use the same random numbers to simulate x_1 and x_2, and if x_1 and x_2 are positively correlated, then the variance of θ is reduced.

On the other hand if x_1 and x_2 are negatively correlated then we have backfiring [148], because by using the same random numbers the variance of the estimator in fact becomes larger than that by using different random numbers. This problem may arise when we compare different models with induced correlation. Therefore, one should apply this method when there is reason to believe that the measure might be positively correlated with respect to input instances [108].

Nevertheless, the technique of common random numbers gives a constant-factor improvement in simulation efficiency, since the cost of generating random numbers is cut in half for simulating the measure of the second model [108]. It should also be pointed out that the variance of estimating the measure of each model is however not affected by the use of this technique.

2.5.2 Control variate technique

The previous section showed that common random numbers can be used to reduce the variance of the difference measure of models. A similar approach can also be applied to a single model. The application of common random numbers to a single model is known as the control variate technique since it introduces the use of control variates. If we need to estimate a measure of a given model or function, we can use another simpler function or model from which we know exactly the value of the corresponding measure. In this case, the measure of such a simpler model or function is defined as the control variate of the original model or function.

To illustrate this technique, let x be an output random variable, and suppose that we need to estimate its mean value, $\mu(x)$. Now let y be another random variable which is the output of a similar, but simplified model. The simplified model is chosen such that the analytical solution of the mean value of y is known. Furthermore, if common random numbers are used for both models, y becomes positively correlated with x. We can then

write an unbiased estimator of $\mu(x)$ as follows

$$\theta = E(x - \alpha(y - E(y)),$$

where $E(\cdot)$ denotes the expected operator. For any real number α, the variance of the estimator becomes

$$\sigma^2(\theta) = \sigma^2(x) + \alpha^2 \sigma^2(y) - 2 \alpha \sigma^2(x, y).$$

It is clear that while the variance of x is constant, y controls the variance of the estimator. Therefore, as long as x and y are positively correlated and the value of $2 \alpha \sigma^2(x, y) > \alpha^2 \sigma^2(y)$, the variance of θ is reduced.

2.5.3 Antithetic variates technique

Another correlation technique to reduce the variance of the estimator for a single model is the antithetic variates method. This practically simple technique is due to Hammersley and Morton [64]. By using two strong negatively correlated unbiased estimators, Y_1, Y_2, for an unknown measure or output, I, the variance of its estimator can be reduced according to the following equation:

$$\sigma^2\left(\frac{1}{2}(Y_1 + Y_2)\right) = \frac{1}{4}\sigma^2(Y_1) + \frac{1}{4}\sigma^2(Y_2) + \frac{1}{2}\sigma^2(Y_1, Y_2). \tag{2.14}$$

where $\sigma^2(\cdot)$ and $\sigma^2(\cdot, \cdot)$ are variance and covariance functions respectively. Thus equation (2.14) provides a criterion for choosing Y_1 and Y_2 which will decrease the variance of estimation.

As an example, let I be the value of a one-dimensional integration of function $g(x)$ from 0 to 1,

$$I = \int_0^1 g(x)\, dx.$$

One can combine the Monte Carlo method and the antithetic variates technique to produce an unbiased estimator of I as follows [138]:

$$\hat{I} = \frac{1}{2N} \sum_{i=0}^{N-1} [g(U_i) + g(1 - U_i)].$$

where U_i denotes a uniform random number, and N is the number of sampling points.

It should be pointed out that for more than one dimensional problems, we have several options for defining Y_2 which is the antithetic variate of Y_1. For example, to estimate the value of two dimensional integration,

$$I = \int_0^1 \int_0^1 g(x, y)\, dx\, dy,$$

we can use the following equation

$$\hat{I} = \frac{1}{4N} \sum_{i=0}^{N-1} [g(U_{2i}, U_{2i+1}) + g(1 - U_{2i}, U_{2i+1}) +$$

$$g(U_{2i}, 1 - U_{2i+1}) + g(1 - U_{2i}, 1 - U_{2i+1})] .$$

The effectiveness of using all or some of the mirror images (antithetic variates) of Y_1 depends on the nature of the integrand and should be investigated in the future. An example for the application of the multi-dimensional antithetic variates method can be found elsewhere [55]. Since it is not known beforehand how great a reduction in variance might be achieved by this technique [148], we may use only one mirror image of Y_1, and the estimator for our example may take the following form

$$\hat{I} = \frac{1}{2\,N} \sum_{i=0}^{N-1} [g(U_{2i}, U_{2i+1}) + g(1 - U_{2i}, 1 - U_{2i+1})] .$$

2.5.4 Conditional expectation method

Sometimes it is possible to transform a complex model into a much simpler model by embedding the original model in a much larger problem. This approach can also be applied for both simulating a complex model and reducing the variance of the model estimator. Such an approach is known as the conditional expectation method. As an example, suppose that θ is an unbiased estimator for the output random variable x, and there is another variate y, for which the mean value of θ, given y, is a known function of y. If we write the random variable z to be this known function

$$z = E_\theta(\theta|y) = g(y),$$

where $E_r(\cdot)$ denotes the expectation operator of random variable r, then z is an unbiased estimator of x

$$E(z) = E_y(E_\theta(\theta|y)) = E_\theta(\theta) \equiv E_x(x),$$

and the variance of z is less than the variance of θ,

$$\begin{aligned} \sigma^2(z) &= \sigma^2(E_\theta(\theta|y)) \\ &= \sigma^2(\theta) - E_y(\sigma^2(\theta|y)), \end{aligned}$$

whenever $E_y(\sigma^2(\theta|y))$ is positive.

2.5.5 Importance sampling method

It has been mentioned previously that the third way to reduce the variance of an estimator is to use a weighted sampling scheme based on prior qualitative or quantitative information. The importance sampling method implements this idea by concentrating the distribution of the sample points in the parts of the region of sampling that are of most "importance" instead of spreading them out evenly [138].

To illustrate the method, let us consider the problem of estimating the integral,

$$I = \int_D g(x)\, dx.$$

The problem can be rewritten in the following form

$$I = \int_D \frac{g(x)}{f(x)} f(x)\, dx,$$

where $f(x) > 0$ for each $x \in D$. The optimal choice of $f(x)$ is [138]

$$f^O(x) = \frac{g(x)}{I},$$

which clearly shows that whenever g(x) is not constant over the region D, the optimal sampling points are not uniformly distributed over the region. In other words, we can reduce the variance of estimating I by actually sampling the points according to the form of $g(x)$. In practice, if it is difficult to generate sampling points from the $g(x)$ distribution, we may use sampling points from an approximation of $g(x)$.

2.5.6 Stratified sampling method

The idea of this method is similar to the idea of importance sampling. In this method, we sample more points in the parts or sampling region that are more "important", but the effect of reducing the variance of the estimator is obtained by concentrating more samples in more important subsets of the region, rather than by choosing the optimal distribution function [138].

To use this method, we first break the sampling region into several disjoint subregions. After that, we sample the points form each subregion according to the distribution over that subregion. In practice, the number of points that we sample for each subregion is proportional to the probability for that subregion. Consequently, if we discretize the region into m of equal probability subregions, which means that the probability of each subregion is $P_i = \frac{1}{m}$, then the number of points we must sample from each subregion will be $N_i = \frac{N}{m}$, where N is the total number of sampling points we need. This particular case of discretization is called systematic sampling or standard Latin hypercube sampling method.

The procedure for this specific version of stratified sampling is as follows [138, 112]:

1. Divide the range $[0, 1]$ of the cummulative distribution $F_x(x)$ into m intervals each of width $\frac{1}{m}$.
2. Generate $\left\{u_{k_i}, k_i = 1, \ldots, \frac{N}{m}; i = 1, \ldots, m\right\}$ from uniform $[0, 1]$.
3. $y_{k_i} \leftarrow \frac{i-1+u_{k_i}}{m}; k_i = 1, \ldots, \frac{N}{m}; i = 1, \ldots, m.$
4. Return $x_{k_i} \leftarrow F_x^{-1}(y_{k_i})$.

An alternative to the standard Latin hypercube sampling method is the midpoint Latin hypercube sampling method [112]. In this case, we choose the median of each subregion to be our sample points. The order of sampling points is then taken randomly. In general, the midpoint rule performs considerably better than the standard rule, however, it may perform worse for a rather subtle, if rare, problem. As an example, if the uncertain inputs of a model are periodic with a wavelength comparable to that induced by the midpoint rule, the midpoint Latin hypercube sampling method could produce misleading results [112].

Chapter 3

Parametric Uncertainty Representations

Before we are able to conduct uncertainty analyses for a given engineering system or model, we need to choose the proper type of representation for parametric uncertainties in such a system. This task is crucial since there is a possibility that different representations will yield different conclusions of the analysis. Therefore, it is important that the robustness of uncertainty analysis results can cover different choices, not inside one class of representations only, but across those classes as well.

For some well-defined problems, the analysis of robustness over classes of representations becomes unnecessary, since there is only one possible correct representation of parametric uncertainty. On the other hand, many practical problems may allow different types of representation, then it is necessary to see whether the change in the type of parametric uncertainty representation will change the results of analysis. In addition to that, both problems in fact still need the in-the-class robustness study for each valid class of representation.

There are four most common uncertainty representations available nowadays: interval mathematics approach, fuzzy theory, Dempster-Shafer theory, and the probabilistic approach. The choice between these classes is the sole responsibility of the analysts. Unfortunately, there is no clear guidance on how to pick one from the others, and the criteria for choosing a specific representation may come from the simplicity and accuracy needed from the representation, or the types of sources of uncertainty. Furthermore, for some systems, it is possible that a combination of these classes of representation is required. This chapter will, therefore, describe these classes of parametric uncertainty representations and some techniques for characterizing uncertain parameters. The intention of these simple descriptions, however, is to give an insight to the analysts when choosing the most appropriate parametric uncertainty representation for the problem of interest.

It should be pointed out that since the classical mathematical tool for characterizing situations under uncertainty has been probability theory, other newer types of representation of parametric uncertainty are always compared and connected with the probabilistic approach. The probabilistic approach is also well in place, and has a lot of theoretical foundations for representing uncertainty. Moreover, it has been proven to be a reliable method in many practical problems [112]. Based on these facts (see Spiegelhalter's paper [157] and references therein for a more detailed discussion), this thesis focuses on the use of the probabilistic approach for characterizing parametric uncertainties in a system. The use of other

appropriate approaches, nevertheless, should be considered whenever it is necessary.

3.1 Interval Mathematics

The simplest way to represent parametric uncertainty in a system is by giving a bounded range of values for the uncertain parameters. In the form of interval mathematics, therefore, such uncertain parameters are assumed to be "unknown but bounded", and each of them has upper and lower limits without a probability structure [144]. Every parameter in the system will have an interval number,

$$[p] \;=\; [\underline{p}, \overline{p}]; \quad \underline{p} \le \overline{p}$$

where for uncertain parameters, the value of $\underline{p}$ should be definitely less than $\overline{p}$. But, on the contrary, for precise parameters, $\underline{p}$ should equal to $\overline{p}$.

In order to be able to work with interval numbers, arithmetic procedures for them should first be studied. Arithmetic operations on intervals, $a = [\underline{a}, \overline{a}]$ and $b = [\underline{b}, \overline{b}]$ may be calculated explicitly as [4]

$$\begin{aligned}
a \;+\; b &= [\underline{a} + \underline{b}, \overline{a} + \overline{b}], \\
a \;-\; b &= [\underline{a} - \overline{b}, \overline{a} - \underline{b}] \;=\; a \;+\; [-1, -1] \cdot b, \\
a \cdot b &= [\min\{\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}\}, \max\{\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}\}], \\
a \;/\; b &= [\underline{a}, \overline{a}] \cdot \left[\frac{1}{\overline{b}}, \frac{1}{\underline{b}}\right]; \quad 0 \notin [\underline{b}, \overline{b}].
\end{aligned}$$

From the above formulas and the definition of precise parameters, it is immediately clear that the arithmetic operations for precise parameters can be derived from those of uncertain parameters. Besides such binary operations, we can also define the unary operation as follows,

$$u(a) \;=\; [\min\; u(a), \max\; u(a)],$$

where $u(\cdot)$ is a unary operator, such as $a^k, \exp(a), \sin(a)$, and others. If the function or the unary operator is monotonic with respect to a then the calculation of its interval is straightforward. Unfortunately, many practical functions are non-monotonic, and moreover, the interval evaluation for those functions depends on the choice of expression. The following example shows this problem. Two expressions for a simple function f can be defined as

$$\begin{aligned}
f_1(x, y) &= \frac{x + y}{x}, \\
f_2(x, y) &= 1 + \frac{y}{x}.
\end{aligned}$$

For this example, we can consider that x and y are independent of each other, and are unknown but bounded such that $x = [1, 2]$, and $y = [0, 1]$. From interval arithmetic formulas above, we have

$$f_1(x, y) \;=\; \frac{[1, 2] + [0, 1]}{[1, 2]} \;=\; [\frac{1}{2}, 3],$$

$$f_2(x,y) \;=\; [1,1] + \frac{[0,1]}{[1,2]} \;=\; [1,2].$$

From the results of interval computations, we see that the second expression produces the exact bound for f, but the first one results in a conservative bound. Therefore, one of problems in using the interval mathematics representation for uncertain parameters is the ability to generate a sharp bound or interval for the results. Normally, the approach to producing better bounds has been to rearrange the expression so that each uncertain parameter appears only once [110].

Another problem which may arise is the possibility of undefined interval or bound for some expressions of a function. Again, a simple example can be used to show this problem. A function, $f(x) = \frac{1}{1+x^2}$, with $x = [-2,2]$ can be expressed as

$$f_1(x) \;=\; \frac{1}{1 + x \cdot x}.$$

If we follow that expression, we have

$$\begin{aligned} f_1(x) \;&=\; \frac{1}{[1,1] + [-2,2] \cdot [-2,2]} \\ &=\; \frac{1}{[1,1] + [-4,4]} \;=\; \frac{1}{[-3,3]}, \end{aligned}$$

which is not defined since the interval contains zero in the denominator part. On the other hand, if we use the unary operator for x^2 in that function, we obtain a well-defined interval,

$$\begin{aligned} f_2(x) \;&=\; \frac{[1,1]}{[1,1] + [-2,2]^2} \\ &=\; \frac{[1,1]}{[1,1] + [0,4]} \;=\; [\frac{1}{5}, 1]. \end{aligned}$$

The problem of undefined interval evaluation in this example actually comes from the independence of uncertain parameters. The first expression, $f_1(x)$, treats $x^2 = x \cdot x$ as like $x \cdot y$, where x and y are independent of each other. On the contrary, the second expression, $f_2(x)$, treats the dependence of $x \cdot x$ correctly. In fact, the problem of conservative bounds described in the first example also arises from a false treatment of dependency in binary operations. Therefore, the approach to rearrange the expression, mentioned earlier, actually works by reducing the degree of dependency of uncertain parameters in the expression. From these two simple examples, we can see that a naive application of interval mathematics may yield results which are not representative as solutions of real problems with uncertain data. Further examples of similar problems have been shown by Kutscher and Schulze [89].

Fichtner, Reinhart, and Rippin [46] showed an example of the application of interval mathematics to designing flexible chemical plants. Unfortunately, they did not discuss the sharpness of the result from the second stage optimization which actually is crucial for solving the first stage optimization. Another example of interval analysis which is given by Veliov [179] deals with parametric and functional uncertainties in dynamic systems. In this example, the uncertain parameter may vary its value at prescribed moments of time within a known bound, and this type of uncertainty is defined as functional uncertainty in contrast to the ordinary case where uncertainty is connected with the value of constant parameters. An interesting example for application of interval mathematics to the development of parameter

sets satisfying the bounded-error constraints has been shown by Moore [111]. The problem in that paper is to estimate the set of all parameters of a model for which the output errors are within given bounds. Given the required tolerance of computation, the methodology can generate boundaries of the regions containing all feasible parameters.

As the programming languages for interval arithmetic become mature [63, 187, 91], the use of interval mathematics for representing parametric uncertainty can be more encouraging. Similarly, the development of robust methodologies to generate sharper bounds on the ranges of values of functions, such as the use of sub-boxes for uncertain parameters [133] or the use of combinatorial interval analysis [40], will also help increasing the number of applications of the interval mathematics approach to uncertainty analysis.

To end this section, an interesting remark from Kutscher and Schulze [89] regarding to the use of interval mathematics to describe uncertainties is quoted below.

> The formal substitution of model parameters with intervals to describe their uncertainties may be compared to the famous method of the ancient Greek figure Procrustes who cut off his guests' legs or stretched their bodies to make them confirm to the length of his bed: The real uncertainties are forced into an interval bed.

3.2 Fuzzy Theory

The notion of fuzzy theory for representing parametric uncertainty comes from the concept of a fuzzy set. A fuzzy set is defined as "a class of objects with a continuum of grades of membership" [197]. On the other hand, the classical set theory uses a crisp set definition by which elements in a given universe are divided into two groups: members and nonmembers [82]. Therefore, the crisp set can be considered to be a restricted case of the more general fuzzy set. Based on this notion of extension, all properties and relationships from the classical set theory are readily stretched, for example, fuzzy logic, fuzzy relations, fuzzy measures, and others.

Another extension is the concept of a fuzzy number [78], and it is defined as "a fuzzy subset of real numbers that is convex and normal". Since a fuzzy number has a continuum of grades of membership, it can be characterized by using a membership function. An ordinary number, A, in the universe of real numbers, R, is defined by its characteristic function, $\mu_A(\cdot)$,

$$\mu_A(x) \in \{0,1\}; \quad \forall\, x \in R,$$

which shows that an element of R belongs to, or does not belong to A according to the value of the characteristic function. In other words, if an element of R, for example y, equals to A then $\mu_A(y) = 1$, otherwise $\mu_A(y) = 0$. Similarly, a fuzzy number, $\mathbf{A}$ can also be defined by its characteristic or membership function, however, this function can take its values in the interval $[0,1]$ instead of in the binary set $\{0,1\}$.

$$\mu_{\mathbf{A}}(x) \in [0,1]; \quad \forall\, x \in R,$$

which means that the elements of R belong to $\mathbf{A}$ with a level located in $[0,1]$. Figure 3-1 shows a membership function of a fuzzy number. For each value of $\alpha \in [0,1]$, there is an interval, $[\underline{A}_\alpha, \overline{A}_\alpha]$, associated with it. This interval should be a closed interval in order to satisfy the convexity property of fuzzy numbers. As we can see, for a certain value of

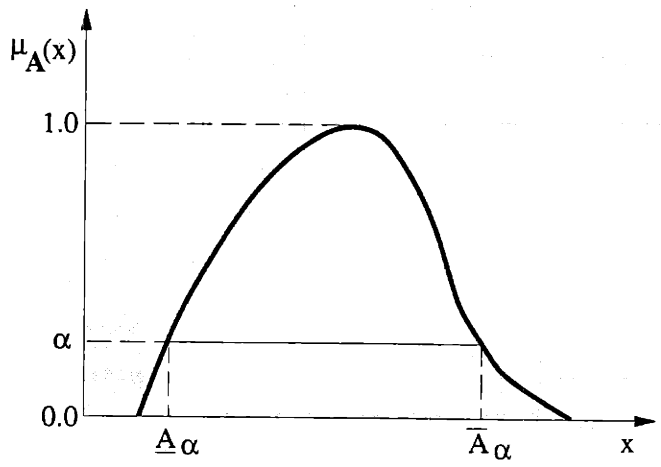


Figure 3-1: Definition of fuzzy numbers

α, the corresponding fuzzy number is actually an interval of numbers. It means that all arithmetic operations for interval mathematics can be used.

Similar to the concept of interval numbers described in the previous section, the concept of fuzzy number requires an appropriate set of arithmetic operations. This required set of operations can be derived from the extension principle proposed by Zadeh [198]. The extension principle is defined in the following.

Definition 3.1 *Let f be a mapping from $U_1 \times U_2 \times \cdots \times U_N$ to a universal set Y such that $y = f(u_1, u_2, \ldots, u_N)$. Given fuzzy numbers $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N$ are in $U_1, U_2, \ldots, U_N$ with membership functions $\mu_{\mathbf{A}_1}, \mu_{\mathbf{A}_2}, \ldots, \mu_{\mathbf{A}_N}$ respectively. The membership function of the fuzzy number $\mathbf{B} = f(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N)$ is defined as:*

$$\mu_{\mathbf{B}}(y) = \begin{cases} \sup\limits_{\substack{u_1,\ldots,u_N \\ y=f(u_1,\ldots,u_N)}} \min(\mu_{\mathbf{A}_1}(u_1), \ldots, \mu_{\mathbf{A}_N}(u_N)), & \text{if } f^{-1}(y) \neq \emptyset, \\ 0, & \text{if } f^{-1}(y) = \emptyset. \end{cases}$$

Another approach for deriving the arithmetic operations on fuzzy numbers is the α-cuts approach [78]. The difference between this approach and the previous one is that the working domain for the α-cuts approach is the membership function, on the contrary, the working domain for the extension principle is the support of fuzzy numbers. As a simple illustration of such a difference, suppose that f is a function mapping a pair of integer numbers from $U_1 = \{0, 1, 2\}$ and $U_2 = \{1, 2\}$ to $Y = \{1, 2, 3, 4\}$. This function can also be written as $f(u_1, u_2) = u_1 + u_2$; $u_1 \in U_1$, $u_2 \in U_2$. Now, let A_1 be a fuzzy number defined on U_1 and let A_2 be a fuzzy number defined on U_2 with membership functions

$$\mu_{\mathbf{A}_1}(u_1) = \begin{cases} 0.5, & u_1 = 0 \\ 1.0, & u_1 = 1 \\ 0.25, & u_1 = 2 \end{cases},$$

and

$$\mu_{\mathbf{A}_2}(u_2) = \begin{cases} 0.5, & u_1 = 1 \\ 1.0, & u_1 = 2 \end{cases}.$$

By using the extension principle for these discrete fuzzy numbers, the membership function of $\mathbf{B}$ becomes

$$\begin{aligned}
\mu_{\mathbf{B}}(y \leq 0) &= 0 \\
\mu_{\mathbf{B}}(y = 1) &= \max(\min(0.5, 0.5)) = 0.5 \\
\mu_{\mathbf{B}}(y = 2) &= \max(\min(0.5, 1.0), \min(1.0, 0.5)) = 0.5 \\
\mu_{\mathbf{B}}(y = 3) &= \max(\min(1.0, 1.0), \min(0.25, 0.5)) = 1.0 \\
\mu_{\mathbf{B}}(y = 4) &= \max(\min(0.25, 0.5)) = 0.25 \\
\mu_{\mathbf{B}}(y \geq 5) &= 0
\end{aligned}$$

Since the membership function for $\mathbf{A}_1$ and $\mathbf{A}_2$ has three different levels, $\{0.25, 0.5, 1.0\}$, then we only have to evaluate α-cuts for these levels to get an exact membership function of $\mathbf{B}$. The interval of $\mathbf{A}_1$ and $\mathbf{A}_2$ for each α-cut can be obtained from the information of their membership functions,

α	$A_{1\alpha}$	$A_{2\alpha}$
0.25	$[0, 2]$	$[1, 2]$
0.5	$[0, 1]$	$[1, 2]$
1.0	$[1, 1]$	$[2, 2]$

By using the interval arithmetic for addition, we have

$$\begin{aligned}
B_{\alpha=0.25} &= [0, 2] + [1, 2] = [1, 4] \\
B_{\alpha=0.5} &= [0, 1] + [1, 2] = [1, 3] \\
B_{\alpha=1.0} &= [1, 1] + [2, 2] = [3, 3],
\end{aligned}$$

and the corresponding membership function is the same as the result of extension principle approach.

Therefore, we can write down these two different approaches for all arithmetic operations. First, the extension principle approach yields the following operations,

$$\begin{aligned}
\mu_{\mathbf{A} \oplus \mathbf{B}}(z) &= \bigvee_{z = x + y} (\mu_{\mathbf{A}}(x) \wedge \mu_{\mathbf{B}}(y)) \\
\mu_{\mathbf{A} \ominus \mathbf{B}}(z) &= \bigvee_{z = x - y} (\mu_{\mathbf{A}}(x) \wedge \mu_{\mathbf{B}}(y)) \\
\mu_{\mathbf{A} \otimes \mathbf{B}}(z) &= \bigvee_{z = x \times y} (\mu_{\mathbf{A}}(x) \wedge \mu_{\mathbf{B}}(y)) \\
\mu_{\mathbf{A} \oslash \mathbf{B}}(z) &= \bigvee_{z = x / y} (\mu_{\mathbf{A}}(x) \wedge \mu_{\mathbf{B}}(y)),
\end{aligned}$$

where $\odot$ denotes the extended arithmetic operation, and $\bigvee$ and $\bigwedge$ represent max and min respectively. Similarly, the α-cuts approach gives the following operations,

$$A_\alpha \oplus B_\alpha = [\underline{A}_\alpha, \overline{A}_\alpha] + [\underline{B}_\alpha, \overline{B}_\alpha]$$

$$
\begin{aligned}
A_\alpha \ominus B_\alpha &= [\underline{A}_\alpha, \overline{A}_\alpha] - [\underline{B}_\alpha, \overline{B}_\alpha] \\
A_\alpha \otimes B_\alpha &= [\underline{A}_\alpha, \overline{A}_\alpha] \times [\underline{B}_\alpha, \overline{B}_\alpha] \\
A_\alpha \oslash B_\alpha &= [\underline{A}_\alpha, \overline{A}_\alpha] \ / \ [\underline{B}_\alpha, \overline{B}_\alpha],
\end{aligned}
$$

which use interval arithmetic operations described in the previous section. It should be pointed out that the difficulties related to the interval mathematics approach will remain for the α-cuts approach. As an example, in terms of obtaining sharper bounds, the use of combinatorial interval analysis may help avoiding the problem of multiple occurence of variables in the algebraic expression [40]. Several methods for calculating functions of fuzzy numbers have been developed, and they are based on the α-cuts approach [40, 196].

In general, we can use fuzzy numbers to represent parametric uncertainties in a system. First, we need to define the range of variability for uncertain parameters, then point out the most possible values and a possibility distribution for other values within a given range. Mathematically, we need to define the support of each uncertain parameter, and its form of membership function. An example for constructing the membership function from non-precise observations was given by Viertl [182]. In this type of construction, the approach is similar to generating the probability density function from the corresponding cumulative distribution function of a random variable. However, the membership function only describes the imprecision of one observation, on the other hand, the density function may describe a set of probable observations. The development of rules for obtaining a membership function from a set of probable observations is currently an active research area. These rules are crucial to the use of statistical inference procedures for non-precise data. Examples for the applications of fuzzy theory to represent parametric uncertainty can be found elsewhere [85, 192]

The main attraction for applying fuzzy theory is when the sources of uncertainty come from the impreciseness of actual observation values or vagueness of verbal descriptions [9] (it should be pointed out that the application of a non-probabilistic approach, such as fuzzy theory, to uncertainty analysis is still questionable for many researchers [157, 186]). For these types of parametric uncertainty, one may consider fuzzy theory representations, and perhaps a combination between fuzzy theory as a non-probabilistic approach, and probabilistic approaches may result in a more general representation for parametric uncertainties. Even though in this thesis we restrict ourselves with the probabilistic approach for representing parametric uncertainties, the probabilistic representations used in this thesis, such as polynomial chaos expansions, can easily be combined with fuzzy representations whenever it is necessary. This combination may be achieved, for example, by assigning fuzzy numbers as the coefficients in the polynomial chaos expansion (see chapter 4 for discussions on the polynomial chaos expansion). Several papers and books have also discussed other combinations between different types of uncertainty representations [78, 87, 116, 131, 182, 186, 195].

3.3 Dempster-Shafer Theory

The polemic of uncertainty representations goes not only between probabilistic and non-probabilistic approaches, but further between different probabilistic approaches as well [121, 146, 186]. As an example, the additivity rule from the classical probability theory was extended by Dempster and Shafer, because they believe that this rule is not suitable as a rule for degrees of belief [145]. In classical probability theory, assigning a probability or belief to a variable or proposition implicitly defines the probability or belief to its complement.

On the other hand, Dempster and Shafer argued that when one assigns a belief to a proposition, it might still be possible that the person is unsure of the belief for the proposition's complement. It means that the additivity rule should be relaxed in order to incorporate such a condition. Mathematically, we can rewrite the previous remarks as

$$\begin{aligned} \Pr(A \cup A^c) &= \Pr(A) + \Pr(A^c) \\ \text{Bel}(A \cup A^c) &\geq \text{Bel}(A) + \text{Bel}(A^c), \end{aligned}$$

where $\Pr(\cdot)$ and $\text{Bel}(\cdot)$ denote the probability and belief functions respectively, and superscript c represents the complementary operation. The "greater-than" inequality is clearly a must because $A \cup A^c = \Theta$, and $\text{Bel}(\Theta) = \Pr(\Theta) = 1$, where Θ is a set of distinct and exhaustive possibilities.

Since $\text{Bel}(A^c)$ can take any values regardless of the value of $\text{Bel}(A)$, a complete description of proposition A needs both $\text{Bel}(A^c)$ and $\text{Bel}(A)$. Shafer used the notion of the degree of doubt, or plausibility function to represent $\text{Bel}(A^c)$.

$$\begin{aligned} \text{Dou}(A) &= \text{Bel}(A^c) \\ \text{Pl}(A) &= 1 - \text{Dou}(A), \end{aligned}$$

where $\text{Dou}(\cdot)$ and $\text{Pl}(\cdot)$ are the degree of doubt and the plausibility function respectively. The relationships between belief, probability, and plausibility functions can be written as follows:

$$\text{Bel}(A) \leq \Pr(A) \leq \text{Pl}(A).$$

From these inequalities, belief and plausibility functions are sometimes known as the lower and upper probability functions respectively, however, these functions can not be considered as probability bounds [129]. Therefore, instead of assigning one fixed value for a variable or proposition, the Dempster and Shafer approach yields two values.

To calculate the belief or the plausibility function of a proposition, one must first define the basic probability assignment or support function which maps any subset of the frame of discernment Θ onto the interval $[0, 1]$. The hierarchy of support functions can be written schematically as follows [145],

$$\left\{ \begin{array}{l} \text{simple} \\ \text{support} \\ \text{functions} \end{array} \right\} \subset \left\{ \begin{array}{l} \text{separable} \\ \text{support} \\ \text{functions} \end{array} \right\} \subset \left\{ \begin{array}{l} \text{support} \\ \text{functions} \end{array} \right\} \subset \left\{ \begin{array}{l} \text{belief} \\ \text{functions} \end{array} \right\}.$$

Simple support functions are belief functions which are based on homogeneous evidence. It means that such an evidence affects only a specific subset of the frame of discernment, and this specific subset is called a focal set. Separable support functions include both simple support functions and orthogonal sums of simple support functions. A separable support function has more than one focal set or proposition, and consequently, has more than one simple support function. The combinations between simple support functions are obtained by applying the orthogonal sum operations.

As mentioned earlier, the support for a focal set is a function m which maps an element of the power set of Θ, denoted by A, onto the interval $[0, 1]$. Therefore, a support function

or basic probability assignment can be defined according to the following rules,

$$\begin{aligned} m(\emptyset) &= 0 \\ 0 \leq m(A) &\leq 1 \\ \sum_{A \in 2^{\Theta}} m(A) &= 1, \end{aligned}$$

where $\emptyset$ is the empty set, and 2^{Θ} denotes the power set of Θ. The belief and plausibility functions of a proposition, denoted by B, can therefore be written as follows [195],

$$\begin{aligned} \mathrm{Bel}(B) &= \sum_{A_i \subseteq B} m(A_i), \\ \mathrm{Pl}(B) &= \sum_{A_i \cap B \neq \emptyset} m(A_i). \end{aligned}$$

Similar to the two previous non-probabilistic approaches, we also need to define the arithmetic operations for Dempster-Shafer real variables. Yager gave a theorem to compute the arithmetic operations on Dempster-Shafer structures [195].

Theorem 3.1 *Assume V_1 and V_2 are two variables whose knowledge are represented by two independent Dempster-Shafer structures m_1 and m_2 with focal elements $A_i, i = 1, \ldots, p$ and $B_j, j = 1, \ldots, q$ respectively. Let $\perp$ be any arithmetic operation. Assume $V = V_1 \perp V_2$, then V induces a Dempster-Shafer structure with basic probability assignment m, where for any $A \subset R$, and R is the set of real numbers:*

$$m(A) = \sum_{A_i \perp B_j = A} m_1(A_i) \times m_2(B_j).$$

In the above theorem, the operation $\perp$ is defined as

$$\begin{aligned} C &= A_i \perp B_j \\ &= \{x \perp y \mid x \in A_i, y \in B_j\}. \end{aligned}$$

Clearly, for this case the issue of normalization does not arise because if A and B are non-null $A \perp B$ is always non-null. The normalization process is required when there exists at least one non-zero orthogonal sum from which yields the empty set [77]. We may be able to directly calculate the belief function of a combination of Dempster-Shafer variables by using an algorithm developed by Silva and Milidiú [150].

In general, we can use Dempster-Shafer structures to represent parametric uncertainties. First, we need to generate focal elements for these uncertain parameters, and assign values to the corresponding simple support functions. After that, the frame of discernment for uncertain parameters and uncertain variables in the model should be developed. The difficulty for generating such a frame of discernment will arise when the number of focal elements is quite large. A similar problem may as well come when we have to deal with continuous uncertain parameters.

The main objection to the use of Dempster-Shafer structures to represent uncertainty arises from its incoherence property [186]. However, the notion of lower and upper probability functions developed along with this theory is indeed useful and practical. This idea generates further developments of imprecise probability theory [186] and the interval probability concept [167]. They currently become the active areas of research for extending

the concept of classical probability theory to incorporate the pluralist viewpoint of uncertainty [129].

3.4 Probabilistic Approaches

As mentioned earlier, this thesis assumes that the probabilistic approach is sufficient for representing classes of problems from chemical and environmental engineering systems in the presence of uncertainty. This probabilistic approach emphasizes the use of random variables, processes, and fields to represent parametric uncertainty. It is clear that the probabilistic approach is the most appropriate tool for representing probabilistic uncertainty, such as stochastic disturbances, variability conditions, risk incorporations, and many others. Sometimes, the uncertainty which arises from the lack of knowledge for the true value of deterministic parameters can also be represented by the probabilistic approach. However, this kind of application requires more attention in the modeling process, and more careful judgement when interpreting the results.

Since random variables, processes, and fields play major roles in the probabilistic approach, we need to clearly define them, and also discuss their main properties (see references [120, 164] for more detailed explanations).

Definition 3.2 *A real-valued random variable, $\mathbf{x}(\omega)$, is a function which maps the probability space Ω into the real line, such that:*

1. *The set $\{\omega \in \Omega \mid \mathbf{x}(\omega) \leq x\}$ is an event for any real number x.*
2. *The probabilities of the events $\{\mathbf{x}(\omega) = +\infty\}$ and $\{\mathbf{x}(\omega) = -\infty\}$ equal zero.*

$$\begin{aligned} \Pr(\mathbf{x}(\omega) = +\infty) &= 0 \\ \Pr(\mathbf{x}(\omega) = -\infty) &= 0 \end{aligned}$$

From the above definition, we can see that every random variable corresponds to a specific probability distribution function. As an example, a continuous real-valued random variable $\mathbf{x}(\omega)$ has a probability density function, $f_x(x)$, if for any Borel set A from Ω, the following equation is valid,

$$\Pr(\mathbf{x}(\omega) \mid \omega \in A) = \int_A f_x(x)\, dx. \tag{3.1}$$

A less abstract relationship between probability and probability density function can be derived from the definition of the cumulative distribution function, $F_x(x)$,

$$F_x(x) = \Pr(\mathbf{x}(\omega) \leq x) = \int_{-\infty}^{x} f_x(u)\, du, \quad x \in R. \tag{3.2}$$

In other words, $\frac{dF_x(x)}{dx} = f_x(x)$ for almost everywhere (a.e.). The previous definition and discussion still hold for a discrete real-valued random variable by changing every integration process with a summation over all possibilities. It is also possible that we have a random variable with mixed type.

From the earlier discussion, we can see that a random variable can be characterized from its probability density function. Other ways to characterize a random variable come from the concept of moments values, the notion of characteristic functions, and many other

transforms of the probability density function (see the next chapter). The definitions of moments, and the characteristic function of a random variable are given below.

Definition 3.3 *The i-th moment of a continuous real-valued random variable, $\mathbf{x}(\omega)$, with probability density function $f_x(x)$ is defined by*

$$m_i(\mathbf{x}(\omega)) = \int_{-\infty}^{+\infty} x^i \, f_x(x) \, dx.$$

These values of moments may define some properties of the random variable. As an example, the first moment, $m_1(\mathbf{x})$, which is also known as the mean or the expected value, $\mu(\mathbf{x})$, defines the central mass of the probability density function, or the average value for the random variable. Similar to the concept of moment in engineering mechanics, we can also have moments of a random variable based on a non-zero point of reference,

$$m_i(\mathbf{x} - p) = \int_{-\infty}^{+\infty} (x - p)^i \, f_x(x) \, dx,$$

where p is a non-zero real number. If we set point p to be the mean value, we then have central moments,

$$c_i(\mathbf{x}) = m_i\left(\mathbf{x} - \mu(\mathbf{x})\right).$$

It is immaterial whether we know the zero point reference moments or the non-zero point reference moments, since we can find immediately that [184]

$$m_i(\mathbf{x} - p) = m_i(\mathbf{x}) - \binom{i}{1} p \, m_{i-1}(\mathbf{x}) + \binom{i}{2} p^2 \, m_{i-2}(\mathbf{x}) + \cdots + (-1)^i \, p^i \, m_0(\mathbf{x}),$$

and vice versa

$$m_i(\mathbf{x}) = m_i(\mathbf{x} - p) + \binom{i}{1} p \, m_{i-1}(\mathbf{x} - p) + \binom{i}{2} p^2 \, m_{i-2}(\mathbf{x} - p) + \cdots + p^i \, m_0(\mathbf{x} - p).$$

Several measures for a random variable that are based on its second, third, and fourth central moments are known to be important. The second central moment, $c_2(\mathbf{x})$, defines the degree of dispersion of the corresponding random variable from its mean value. It is also called the variance denoted by $\sigma^2(\mathbf{x})$, and its square root is known as the standard deviation. The third central moment affects the degree of asymmetry of the corresponding probability density function around its mean value. A coefficient known as the skewness factor is used to measure this degree of asymmetry, and it is defined by

$$\gamma_1(\mathbf{x}) = \frac{c_3(\mathbf{x})}{\sigma^3(\mathbf{x})}.$$

A positive value of skewness factor relates to a probability density function with a more tail area towards more positive x; a negative value signifies a probability density function with a more tail area towards more negative x; a zero value means a symmetrical probability density function. Another important non-dimensionalized coefficient is the kurtosis factor. It measures the relative peakedness or flatness of a probability density function with respect

to the Gaussian probability density function, and it can be derived from the fourth central moment,

$$\gamma_2(\mathbf{x}) = \frac{c_4(\mathbf{x})}{\sigma^4(\mathbf{x})}.$$

The kurtosis factor for a Gaussian probability density function equals to three. Therefore, a less-than-three kurtosis factor defines a less peaknedness or more flatness probability density function than the Gaussian; a more-than-three kurtosis factor means a more peaknedness or less flatness probability density function than the Gaussian. Platykurtic and leptokurtic are other names for probability density functions with a less-than-three and a more-than-three kurtosis factors respectively.

It is often more convenient to calculate moments of random variable $\mathbf{x}$, and in turn the corresponding central moments, by using its characteristic function, $\Phi_x(\cdot)$, or by using its moment generating function, $\Psi_x(\cdot)$. A random variable with probability density function $f_x(x)$ has characteristic and moment generating functions

$$\begin{aligned}\Phi_x(u) &= \int_R \exp(iux)\ f_x(x)\ dx,\\ \Psi_x(s) &= \int_R \exp(sx)\ f_x(x)\ dx,\end{aligned}$$

where u and s are arbitrary real numbers. It is obvious that the probability density function of a random variable and its characteristic function form a pair of Fourier transforms,

$$f_x(x) = \frac{1}{2\pi}\int_R \exp(-iux)\ \Phi_x(u)\ du.$$

The MacLaurin series of $\Phi_x(u)$ relates the characteristic function with moments of the corresponding random variable [164],

$$\begin{aligned}\Phi_x(0) &= m_0(\mathbf{x}),\\ \left.\frac{d\Phi_x(u)}{du}\right|_{u=0} &= i\ m_1(\mathbf{x}),\\ &\vdots\\ \left.\frac{d^n\Phi_x(u)}{du^n}\right|_{u=0} &= i^n\ m_n(\mathbf{x}).\end{aligned}$$

Similarly for the moment generating function, we have

$$\begin{aligned}\Psi_x(0) &= m_0(\mathbf{x}),\\ \left.\frac{d\Psi_x(s)}{ds}\right|_{s=0} &= m_1(\mathbf{x}),\\ &\vdots\\ \left.\frac{d^n\Psi_x(s)}{ds^n}\right|_{s=0} &= m_n(\mathbf{x}).\end{aligned}$$

All concepts for a random variable shown previously can be extended to vector-valued random variables. The extension of those concepts will induce several new terms or defini-

tions, such as the joint cumulative distribution function, the marginal probability density function, independent random variables, and many others.

Definition 3.4 *The joint cumulative distribution function of a sequence of n continuous real-valued random variables,* $\{\mathbf{x_n}\}$ *is defined by*

$$F_{x_1 x_2 \cdots x_n}(x_1, x_2, \ldots, x_n) \;=\; \Pr\left((\mathbf{x_1} \leq x_1) \cap (\mathbf{x_2} \leq x_2) \cap \cdots \cap (\mathbf{x_n} \leq x_n)\right),$$

and the corresponding joint probability density function becomes

$$f_{x_1 x_2 \cdots x_n}(x_1, x_2, \ldots, x_n) \;=\; \frac{\partial^n F_{x_1 x_2 \cdots x_n}(x_1, x_2, \ldots, x_n)}{\partial x_1 \, \partial x_2 \, \cdots \, \partial x_n},$$

if this partial derivative exists.

In the context of a set of random variables, $\{\mathbf{x_n}\}$, the probability density function of each element of that set, $\mathbf{x_i}$, is usually called the marginal probability density function, and it is defined as follows,

$$f_{x_i}(x_i) \;=\; \int_{R^{n-1}} f_{x_1 x_2 \cdots x_n}(x_1, x_2, \ldots, x_n)\; dx_1\; dx_2\; \cdots\; dx_{i-1}\; dx_{i+1}\; \cdots\; dx_n.$$

If the joint probability density function of a set of random variables can be written as a product of marginal probability density functions of its element, then that set of random variables is independent or mutually independent. There is a similar concept for a pair of random variables, it is called the correlation concept. Two random variables $\mathbf{x}$ and $\mathbf{y}$ are uncorrelated if the following equation is satisfied,

$$m_{1,1}(\mathbf{x}, \mathbf{y}) \;\equiv\; \int_{R^2} x\; y\; f_{xy}(x, y)\; dx\; dy \;=\; m_1(x)\; m_1(y),$$

where $m_{i,j}(\mathbf{x}, \mathbf{y})$ is the i, j-th joint moment of $\mathbf{x}$ and $\mathbf{y}$. It should be noted that pairwise independence or correlation of elements in a set of random variables does not imply independence of the total set. Another pairwise feature is the orthogonalization concept. Two random variables are orthogonal each other if the expectation of their product is zero,

$$m_{1,1}(\mathbf{x}, \mathbf{y}) \;=\; 0.$$

After discussing random variables and their properties, the next step is to browse concepts and classifications of random processes and fields. To clarify the concept, it is important to note that other names for random processes which can be found in different books, are stochastic processes and random functions. By definition, a random process can be considered as a random variable with one parameter, and a random field, which is the extension of a random process, can be as well considered as a random variable with several parameters. We shall use the notation $\mathbf{x}(\omega, t)$ to represent a random process, where t denotes the parameter. Similarly, a random field can be written as $\mathbf{x}(\omega, \underline{s})$, where $\underline{s}$ represents a vector of parameters. We can, therefore, classify a random process based on the continuity of parameter in that process. If the parameter of a random process is continuous, the process is called continuous-parameter random process, as an example, when its parameter is the time or a spatial variable. Clearly, the other case is the discrete-parameter random process. A mixed type of parameters, therefore, can occur in a random field.

A random process, $\mathbf{x}(\omega, t)$, can have several interpretations [120]:

1. It is a family, or an ensemble, of functions $\mathbf{x}(\omega, t)$. It means that, both ω and t are considered as variables.

2. It is a single function of t, or a sample of the given process. In this representation, t is a variable and ω is fixed.

3. It is a random variable. This happens when ω is the variable and t is fixed.

4. It is a real number. In this case, both t and ω are fixed.

There are two different types of random processes based on the property of a single realization of a random process. The first one is called the regular random process. It consists of a family of functions that cannot be described in terms of a finite number of parameters. An example of a regular random process is the Brownian motion or the microscopic motion of particles in collision with molecules in a fluid. The future of a realization or a sample of such a process cannot be determined in terms of its past. The second type of random processes is the predictable random process. An example for this type is

$$\mathbf{x}(\omega, t) = k(\omega)\sin(l(\omega) + t),$$

such that a realization of this process, $\mathbf{x}(\cdot, t)$, is completely determined by the realization $k(\cdot)$ and $l(\cdot)$. In other words, if we know the values of $\mathbf{x}(\cdot, t)$ for $t \leq t_0$, then we can determine the values of $\mathbf{x}(\cdot, t)$ for $t > t_0$.

At a specific value of $t \in T$, a random process becomes a random variable. It also means that a random process is a family of random variables, $\mathbf{x}(\omega, t_1), \mathbf{x}(\omega, t_2), \ldots$, depending upon the set T. Therefore, in the case of the discrete-parameter random process, that family of random variables is finite or countably infinite, and obviously for the case of the continuous-parameter random process, the number of random variables becomes noncountably infinite. Based on this point of view, a random process can have many orders of distribution. The distribution of a random process at a specific value of t is called as the first-order distribution function. Furthermore, the joint distribution function of a random process at two different values of t is known as the second-order distribution function, and we can extend this argument for describing higher order distribution functions.

An important measure derived from the second-order distribution function of a random process is the correlation function. The joint moment of a random process, $\mathbf{x}(\omega, t)$, at two different values of its parameter, t_1, t_2, measures the linear dependence between $\mathbf{x}(\omega, t_1)$ and $\mathbf{x}(\omega, t_2)$. This joint moment is known as the auto-correlation function denoted by $\Gamma_{xx}(t_1, t_2)$. The term cross-correlation $\Gamma_{xy}(t_1, t_2)$ describes the dependence between two random processes, one at t_1, and the other at t_2.

Since a random process can have a collection of distribution functions, we can then classify a random process from the statistical regularity of its collection of distribution functions. It can belong to the class of stationary, or the class of non-stationary stochastic processes.

Definition 3.5 *A random process, $\mathbf{x}(\omega, t)$; $t \in T$, is said to be stationary or strictly stationary if its collection of probability density function stays invariant under an arbitrary translation of its parameter t, that is, for each n, for an arbitrary value of t, we have*

$$F_{x_1 x_2 \cdots x_n}(x_1, t_1; x_2, t_2; \ldots; x_n, t_n) = F_{x_1 x_2 \cdots x_n}(x_1, t_1 + \tau; x_2, t_2 + \tau; \ldots; x_n, t_n + \tau);$$
$$t_j + \tau \in T, \quad j = 1, 2, \ldots, n.$$

It means that the statistical properties of a stationary random process are independent of the origin value of its parameter t. As an example, the mean value of a stationary random process, $\mu(\mathbf{x}(\omega,t))$ is constant, and its auto-correlation function $\Gamma_{xx}(t,t+\tau)$ becomes a function of the difference parameter τ, that is, $\Gamma_{xx}(\tau)$.

Practically, it is difficult to assess a random process whether it holds the stationary requirement for all n. A relaxed version of the stationary random process, such as the wide-sense stationary process, can then be applied in this situation.

Definition 3.6 *A random process, $\mathbf{x}(\omega,t)$; $t \in T$, is a wide-sense stationary process if*

$$\begin{aligned} |\mu(\mathbf{x}(\omega,t))| &= \text{constant} < \infty \\ \sigma^2(\mathbf{x}(\omega,t)) &< \infty \\ m_{1,1}\left(\mathbf{x}(\omega,t_1),\mathbf{x}(\omega,t_2)\right) &= \Gamma_{xx}\left(t_2-t_1\right) \end{aligned}$$

are satisfied.

Other names for the wide-sense stationary process are the second-order, covariance, or weakly stationary process. This definition shows that a strictly stationary random process with finite joint second moment is in fact a wide-sense stationary random process.

Because the joint second moment, or the correlation function, of random processes plays a crucial role in modeling the processes, the Fourier transform for the correlation function is developed. The Wiener-Khintchine formulas describe such a pair of Fourier transforms,

$$\begin{aligned} S_{xx}(\lambda) &= \frac{1}{\pi}\int_{-\infty}^{+\infty} \exp(-i\lambda\tau)\,\Gamma_{xx}(\tau)\,d\tau, \\ \Gamma_{xx}(\tau) &= \frac{1}{2}\int_{-\infty}^{+\infty} \exp(-i\lambda\tau)\,S_{xx}(\lambda)\,d\lambda, \end{aligned}$$

where $S_{xx}(\lambda)$ indicates the power spectral density function of the random process, and it has real and non-negative value. One important result from the Wiener-Khintchine formulas is a simple derivation which shows that the integral of $S_{xx}(\lambda)$ is two times of the second moment of the random process,

$$m_2(\mathbf{x}(\omega,t)) = \Gamma_{xx}(0) = \frac{1}{2}\int_{-\infty}^{+\infty} S_{xx}(\lambda)\,d\lambda.$$

Similar to the use of Fourier transform as another way to characterize a random variable, the power spectral density function can be a tool to classify random processes. As an example, white-noise, which is a wide-sense stationary random process mostly used as an abstraction or limiting process of physical phenomena, has a constant power spectral density function over the whole spectrum, $S_{ww}(\lambda) = S_0;\ -\infty \le \lambda \le +\infty$. Consequently, its correlation function has the following form,

$$\Gamma_{ww}(\tau) = \pi\, S_0\, \delta(\tau),$$

where $\delta(\cdot)$ is the Dirac delta function. This implies that a white-noise is a random process that has no correlation at different values of parameter t.

Random processes are not only categorized according to their statistical regularity described previously, but also based on how its present and future states depend on its past states. Such a dependence is measured by the memory property of the corresponding ran-

dom process. There are two classes of random processes in this case: the class of purely random processes, and the class of Markov random processes. A purely random process has no memory, therefore, at a specific value of its parameter, the process is independent to itself at different values of parameter. Mathematically, we can write the distribution of a purely random process to be a product of its first-order distributions at different values of parameter,

$$F_{x_1 x_2 \cdots x_n}(x_1, t_1; x_2, t_2; \ldots; x_n, t_n) = \prod_{j=1}^{n} F_{x_j}(x_j, t_j); \quad \forall n.$$

Clearly, for a continuous-parameter, a purely random process is not practically realizable, because in the real physical situation, the value of $\mathbf{x}(\omega, t_1)$ always affects the value of $\mathbf{x}(\omega, t_2)$ when t_1 is sufficiently close to t_2. An example for such an abstract random process is the white noise mentioned earlier. While all statistics of a purely random process are generated from the knowledge of its first distribution functions, all statistics for a Markov random process depend on its second-order probability distributions. Since Markov random processes can represent a large number of real stochastic processes, this type of random process has been studied extensively. The formal definition of a Markov random process is given below [120]

Definition 3.7 *A random process, $\mathbf{x}(\omega, t)$; $t \in T$, is a Markov random process, if for every n, and for $t_1 < t_2 < \cdots < t_n$ in T, its conditional distribution has the following form:*

$$F_{x_n | x_{n-1} \cdots x_1}(x_n, t_n \mid x_{n-1}, t_{n-1}; \ldots; x_1, t_1) = F_{x_n | x_{n-1}}(x_n, t_n \mid x_{n-1}, t_{n-1}).$$

This definition implies that a Markov random process is a collection of functions whose conditional probability distribution at a given value of its parameter depends only on its latest past value. The above definition can be generalized to include random processes with higher-order memory. Therefore, a Markov random process of k-th order represents a random process whose statistics are completely specified by its k-th order distribution functions. It means that a purely random process is actually a Markov random process of first order.

Another practically important class of random processes is the class of independent increment random processes. The Wiener-Levy process and the Poisson process are examples of random processes which belong to this class. The Wiener-Levy process also belongs to the class of stationary random processes. Definitions of independent increment and stationary independent increment random processes are given below.

Definition 3.8 *A random process, $\mathbf{x}(\omega, t)$; $t \in T$, is an independent increment random process, if for all $j = 2, \ldots, n$, the increments $\Delta_x(t_i, t_j) = \mathbf{x}(\omega, t_j) - \mathbf{x}(\omega, t_i)$; $i = j - 1$ are mutually independent.*

Definition 3.9 *Consider an independent increment random process, $\mathbf{x}(\omega, t)$; $t \in T$. If the probability distributions of its increments, $\Delta_x(t_1, t_2), \Delta_x(t_2, t_3), \ldots, \Delta_x(t_{n-1}, t_n)$ depend only on the differences of parameters, $t_2 - t_1, t_3 - t_2, \ldots, t_n - t_{n-1}$, then $\mathbf{x}(\omega, t)$ is a stationary independent increment random process.*

All previously mentioned definitions and terms for random processes can be extended to cover random fields. However, the complexity for characterizing a random field is far more than that of random processes, since now we have to deal with a larger number of

parameters. Furthermore, we also need to deal with different names used for defining the same notion. As an example, for a random field, the term "homogeneous" is commonly used instead of the term "stationary". Since a random field has a vector of parameters, we can divide further the class of homogeneous random fields, such as isotropic, ellipsoidal, separable, and direction-dependent homogeneous random fields. More detailed discussions and explanations about random fields are referred to Vanmarcke's book [178].

After describing definitions and terms for random variables, processes, and fields, the next step is to discuss methods for manipulating these quantities. Therefore, similar to other approaches to represent uncertainty, we need a set of arithmetic procedures for random variables, processes, and fields. Since for every random variables, processes, or fields, there are probability functions associated with them, then that set of arithmetic procedures should be derived from the axioms of probability theory. There are three axioms of probability theory:

1. The probability of any event, A, must lie between zero and one,

$$0 \leq \Pr(A) \leq 1.$$

2. The probability of the entire sample space or events, S, is unity,

$$\Pr(S) = 1.$$

3. If the events A and B are mutually exclusive, $A \cap B = \emptyset$, then the probability of their union can be expressed as a sum of their probabilities,

$$\Pr(A \cup B) = \Pr(A) + \Pr(B).$$

These axioms are also valid for conditional probabilities, because conditional probabilities are indeed probabilities. It means that they satisfy the axioms:

$$\begin{aligned}
&\Pr(A \mid B) \geq 0, \\
&\Pr(A \mid A) = 1, \\
&\Pr(A \mid B) + \Pr(A^c \mid B) = 1, \\
&\Pr(A, B \mid C) = \Pr(A \mid B, C)\, \Pr(B \mid C), \\
&\Pr(A \cup B \mid C) = \Pr(A \mid C) + \Pr(B \mid C) - \Pr(A, B \mid C),
\end{aligned}$$

where by definition the conditional probability of event A, assuming that event B occurs is the ratio

$$\Pr(A \mid B) = \frac{\Pr(A, B)}{\Pr(B)},$$

and $\Pr(A, B)$ is another notation for $\Pr(A \cap B)$.

Two important theorems can be obtained from the above relationships of the conditional probability: total probability theorem and Bayes' theorem. The total probability theorem shows how one can infer the total probability of a certain event from the knowledge of a set of conditional probabilities of this event.

Theorem 3.2 *If $\{A_1, A_2, \ldots, A_n\}$ is a partition of the sample space S, and B is an arbi-*

trary event, then

$$\Pr(B) = \sum_{i=1}^{n} \Pr(B \mid A_i)\ \Pr(A_i).$$

On the other hand, Bayes' theorem allows one to relate two conditional probabilities for two events.

Theorem 3.3 *If $\{A_1, A_2, \ldots, A_n\}$ is a partition of the sample space S, and B is an arbitrary event, then*

$$\Pr(A_i \mid B) = \frac{\Pr(B \mid A_i)\ \Pr(A_i)}{\sum_{j=1}^{n} \Pr(B \mid A_j)\ \Pr(A_j)},$$

or using the total probability theorem, we have

$$\Pr(A_i \mid B) = \frac{\Pr(B \mid A_i)\ \Pr(A_i)}{\Pr(B)}.$$

These relationships on probabilities can be extended to the case of probability density functions by the use of equations (3.1) and (3.2).

As mentioned earlier, a set of arithmetic procedures for random variables, process, or fields should be derived from these relationships of probabilities. The aim of these arithmetic procedures is to be able to generate the probability density function of a function of random variables, processes, or fields. Actually, we only need to devise a set of arithmetic procedures for the case of random variables, because by definition, random processes and fields can be considered as random variables with parameters. Simple arithmetic operations involving two independent random variables **A** and **B** are given below.

$$\begin{aligned}
f_{C=A+B}(C) &= \int_{-\infty}^{\infty} f_A(C-B)\ f_B(B)\ dB, \\
f_{C=A-B}(C) &= \int_{-\infty}^{\infty} f_A(C+B)\ f_B(B)\ dB, \\
f_{C=A\times B}(C) &= \int_{-\infty}^{\infty} \frac{1}{B}\ f_A(C/B)\ f_B(B)\ dB, \\
f_{C=A/B}(C) &= \int_{-\infty}^{\infty} B\ f_A(C\times B)\ f_B(B)\ dB.
\end{aligned}$$

For more general cases, we can use theorems in the next chapter which provide the mechanism for generating the probability density function of functions of random variables.

In the probabilistic approach, uncertain parameters and variables in a model or system are represented by random variables, random processes, or random fields. If those uncertain parameters and variables are functions of time or a spatial coordinate, random processes can be used to portray those quantities. Clearly, for uncertain parameters and variables which are functions of time and spatial coordinates, we can use random field representations. The next chapter will focus on different approaches for representing random variables, and then in chapter 5, several techniques for approximating random processes or fields will be given.

3.5 Characterization of Uncertainty

Before we can use the probabilistic approach to perform uncertainty analysis of a model or a system, we first have to assess the input or parametric uncertainties of the model.

What we need is the description of probability distributions of the uncertain parameters or inputs. By knowing the probability distributions of uncertain parameters and inputs, one can apply algebraic manipulation methodologies for random variables, processes, or fields to generate the probability distributions of outputs or variables of interest. In general, this activity involves identification, measurement, or estimation methods.

It should be noted that all identification and estimation methods may be the sources of some errors or uncertainties themselves. It means that we can have several levels of uncertainties while performing the uncertainty analysis, and we have to tackle them one by one. Therefore, the robustness study mentioned in the beginning of this chapter is recommended to handle such a problem.

In this section, we will only deal with one of the major issues in performing uncertainty analyses, that is the characterization of model parameter uncertainty. There are two common methods for characterizing the probability distribution of uncertain parameters or variables in the system: the Bayesian technique, and the maximum entropy approach.

3.5.1 Bayesian Approach

A review article by Stewart, Caracotsios, and Sørensen [161] on the Bayesian approach will be the main reference for this subsection. In that article, they showed a broad over-view of parameter estimation problems from multiresponse data, and described techniques that are based on a famous theorem by Bayes (shown in previous section) on the inference of causes from observed effects, to handle such problems.

The estimation of parameters in a model problem can be posed as follows:

$$y_{i,u} \;=\; g_i(\boldsymbol{x}_u, \boldsymbol{\theta}) + \epsilon_{i,u}; \quad i = 1, \ldots, m; u = 1, \ldots, n,$$

where a model $g_i(\boldsymbol{x}_u, \boldsymbol{\theta})$ for response i at the experimental design point $\boldsymbol{x}_u$ is given, and the problem is to estimate not just the most probable values of the parameters $\boldsymbol{\theta}$, and the covariance of the errors $\epsilon_{i,u}$, denoted by $\boldsymbol{\Sigma}$, but the determination of their joint probability distribution. $y_{i,u}$ represents the value of response i at an independent event or measurement u.

In this problem, the errors are modeled by an m-dimensional Gaussian distribution (see [54] as an example for the use of the Edgeworth series expansion to treat the case of non-Gaussian errors), with expectations $m_1(\epsilon_{i,u}) = 0$, and its covariance matrix, $\boldsymbol{\Sigma}$; $\Sigma_{i,j} = m_{1,1}(\epsilon_{i,u}, \epsilon_{j,u})$, is unknown. Clearly, now we have three uncertain or random variables: the responses $\mathbf{y}$, the parameters, $\boldsymbol{\theta}$, and the covariance matrix, $\boldsymbol{\Sigma}$. Since at a specific value of $\boldsymbol{\theta}$, the model $\{g_i(\boldsymbol{x}_u, \boldsymbol{\theta})\}$ becomes deterministic, and $\{\epsilon_{i,u}\}$ are jointly Normal distributed, then $\{y_{i,u}\}$ are also jointly Normal distributed,

$$f_{y|\theta,\Sigma}(\mathbf{y} \mid \boldsymbol{\theta}, \boldsymbol{\Sigma}) \;\propto\; |\mathbf{A}|^{\frac{n}{2}} \exp\left(-\frac{1}{2} \operatorname{tr}[\mathbf{Av}]\right),$$

where $\mathbf{A} = \boldsymbol{\Sigma}^{-1}$, and $\mathbf{v}$ denotes a symmetric matrix with elements,

$$v_{i,j} \;=\; \sum_{u=1}^{n} \left(y_{i,u} - g_i(\boldsymbol{x}_u, \boldsymbol{\theta})\right) \left(y_{j,u} - g_j(\boldsymbol{x}_u, \boldsymbol{\theta})\right).$$

The likelihood function $l_{\theta,\Sigma|y}(\boldsymbol{\theta}, \boldsymbol{\Sigma} \mid \mathbf{y})$ can be constructed as the function $f_{y|\theta,\Sigma}(\mathbf{y} \mid \boldsymbol{\theta}, \boldsymbol{\Sigma})$ with y now given, whereas $\boldsymbol{\theta}$, and $\boldsymbol{\Sigma}$ are free to vary. According to Bayes' theorem,

multiplication of the likelihood by a suitable prior density, $f_{\theta,\Sigma}(\boldsymbol{\theta}, \boldsymbol{\Sigma})$, gives the posterior density function $f_{\theta,\Sigma|y}(\boldsymbol{\theta}, \boldsymbol{\Sigma} \mid \mathbf{y})$,

$$f_{\theta,\Sigma|y}(\boldsymbol{\theta}, \boldsymbol{\Sigma} \mid \mathbf{y}) \quad \propto \quad l_{\theta,\Sigma|y}(\boldsymbol{\theta}, \boldsymbol{\Sigma} \mid \mathbf{y})\ f_{\theta,\Sigma}(\boldsymbol{\theta}, \boldsymbol{\Sigma}). \tag{3.3}$$

Assuming that $\boldsymbol{\theta}$ and $\boldsymbol{\Sigma}$ are independent from each other, and using the "non-informative" prior distribution of $\boldsymbol{\Sigma}$, and a locally uniform prior distribution for $\boldsymbol{\theta}$ [17], their joint distribution becomes

$$f_{\theta,\Sigma}(\boldsymbol{\theta}, \boldsymbol{\Sigma})\ d\boldsymbol{\theta}\ d\boldsymbol{\Sigma} \quad \propto \quad |A|^{-\frac{m+1}{2}}\ d\boldsymbol{\theta}\ d\boldsymbol{\Sigma}.$$

Combining this prior and the likelihood from equation (3.3), we obtain the following posterior density function,

$$f_{\theta,\Sigma|y}(\boldsymbol{\theta}, \boldsymbol{\Sigma} \mid \mathbf{y}) \quad \propto \quad |\mathbf{A}|^{\frac{n-m-1}{2}}\ \exp\left(-\frac{1}{2}\ \mathrm{tr}\left[\mathbf{Av}\right]\right),$$

and the marginal density function for $\boldsymbol{\theta}$ is calculated by integrating out $\boldsymbol{\Sigma}$ [17],

$$f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \quad \propto \quad |\mathbf{v}|^{-\frac{n}{2}}. \tag{3.4}$$

It should be pointed out that the values of $\boldsymbol{\theta}$ and $\boldsymbol{\Sigma}$ are actually deterministic but unknown. Our lack of knowledge for these values forces us to treat them as random variables. Nevertheless, with all the above assumptions about the errors $\{\epsilon_{i,u}\}$, and with a given set of values for responses, $\{y_{i,u}\}$, we are only able to infer the values of the parameters probabilistically. In other words, we can use $f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})$ as our best knowledge for the values of parameters, and this probability density function is the result from characterizing the uncertainty in the values of the parameters.

Since the parameters and the covariance matrix are in fact deterministic, we may want to search for the posterior density function with the smallest variance. It also means that we want to increase the utility of our information. As an example, Stewart, Caracotsios, and Sørensen [161] showed that a sharper posterior distribution function can be obtained by replacing $\boldsymbol{\Sigma}$ by its conditional maximum-density estimate.

Stewart, Caracotsios, and Sørensen [161] also gave different results for different structures of responses and covariance matrices. The previously shown result (equation (3.4)) is a Type 1 problem, in which we have full responses and a full unknown covariance matrix. Type 2 consists of problems with block-rectangular responses and a block-rectangular covariance matrix, the other types include cases of the irregular responses with a full or partial covariance matrix, and of irregular responses with a heteroscedastic covariance matrix.

To clarify the idea of the Bayesian approach for characterizing parametric uncertainty, we will, in the following, show the use of Monte Carlo simulation techniques in the Bayesian framework for inferring the probability distribution of endogenous stochastic model parameters from measured data (the discussion of other Bayesian computational methods is referred to Smith's paper [153]). We will, therefore, use the Gibbs sampling scheme to efficiently obtain the marginal densities of the posterior distribution, and also apply the "simulated annealing" method for searching the modal position of the posterior distribution. The following discussions will highlight those two methods.

Gibbs Sampling Scheme

The popularity of the Gibbs sampling scheme was started with the paper of Geman and Geman [53] who applied such a scheme to compute an image restoration model. The appealing features of this scheme are the conceptual simplicity and the ease of implementing such an algorithm. We can generate a sample from the joint distribution $f_{x_1 \cdots x_k}(x_1, \ldots, x_k)$, where $\{x_j\}$ is a set of random variables, according to the iterations of the following systematic scan:

$$
\begin{array}{llll}
1. & \text{Sample} \quad x_1^{i+1} & \text{from} & f_{x_1|x_2^i \cdots x_k^i}(x_1 \mid x_2^i, \ldots, x_k^i) \\
2. & \text{Sample} \quad x_2^{i+1} & \text{from} & f_{x_2|x_1^{i+1} x_3^i \cdots x_k^i}(x_2 \mid x_1^{i+1}, x_3^i, \ldots, x_k^i) \\
 & \vdots & & \vdots \\
k. & \text{Sample} \quad x_k^{i+1} & \text{from} & f_{x_k|x_1^{i+1} \cdots x_{k-1}^{i+1}}(x_k \mid x_1^{i+1}, \ldots, x_{k-1}^{i+1})
\end{array}
$$

where superscript i represents the i-th iteration.

It is clear that the implementation of the Gibbs sampling scheme depends on the ability to sample from the conditional distributions at each step. We will use a general approximation method termed as griddy-Gibbs sampler to sample from any conditionals [135]. Since this method does not require the conjugacy property and log-concavity of the conditionals, it is suitable for many practical problems which could be very non-linear and may generate a multi-modal distribution as well.

In many applications of the Gibbs sampler, the conditional distribution is univariate. The idea of the griddy-Gibbs sampler is to make a simple approximation to the inverse cumulative distribution function based on the evaluation of $f_{x_i|x_j}(x_i \mid x_j, j \neq i)$ which is univariate on a grid of points. Therefore, to sample from such a conditional, we can use techniques to generate sampling points from a random variable which has a piece-wise constant probability density function (see chapter 2). The accuracy of this method can be increased by refining and growing the grid adaptively [135].

Since we actually do not generate the posterior distribution, but sample points from it, we can then also efficiently study the sensitivity or robustness of the posterior with respect to the different types of prior distributions. This efficiency is achieved by the use of a technique called the weighted bootstrap or the sampling-resampling method [154].

Simulated Annealing

As mentioned earlier, many practical problems may generate a posterior distribution which is multi-modal in the domain of parameters. Consequently, if we want to search the maximum a posteriori estimator for such a distribution, we have to consider methods for dealing with global optimization issues.

In the spirit of Monte Carlo methods, the "simulated annealing" method can be used to optimize a multi-modal posterior distribution. When the parameters in the model are continuous, we may use a continuous version of the "simulated annealing" algorithm [33]. This algorithm can be written down as follows:

1. Start with a given point $\mathbf{x}_0$, and a high value of parameter T_0 which is called the "temperature"

2. Generate a succession of points: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots$ tending to the global maximum of the distribution. New candidate points are generated around the current point $\mathbf{x}_i$ by

applying random moves along each coordinate direction, in turn. If the point falls outside the domain of the posterior distribution, a new point is randomly generated until a point belonging to this domain is found.

3. A candidate point $\mathbf{x}'$ is accepted or rejected according to the Metropolis criterion:
   If $\Delta pos() \geq 0$, then accept the new point: $\mathbf{x}_{i+1} = \mathbf{x}'$
   else accept the new point with probability:
   $\Pr(\Delta pos()) = \exp(-\frac{\Delta pos()}{T})$
   where $\Delta pos()$ is the change of values of the posterior distribution, and T is a parameter called "temperature" in step 1.

4. After "thermal equilibrium", we record the best point as $\mathbf{x}_{\text{opt}}$. We reduce the "temperature" according the schedule we choose and go back to step 2. This process is terminated when the "temperature" is low enough that no more useful improvement can be expected.

The specific implementation of this algorithm is obtained from the **netlib** server, and it has been modified by Goffe *et al.* It also has been tested against the Nelder and Mead simplex method, and against a version of the Adaptive Random Search method. This method proved to be more reliable than the others, because it was always able to find the optimum or at least a point very close to it [33]. The implementation of this method uses the MZT random number generator mentioned in the previous chapter to generate random moves.

Example 3.5.1.1

We will consider a simple kinetic modeling example taken from the paper by Box and Draper [17]. A system of chemistry, $\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C}$, is modeled by the following set of differential equations:

$$\begin{aligned} \dot{\eta_{\mathbf{A}}} &= -\phi_1 \eta_{\mathbf{A}} \\ \dot{\eta_{\mathbf{B}}} &= \phi_1 \eta_{\mathbf{A}} - \phi_2 \eta_{\mathbf{B}} \\ \dot{\eta_{\mathbf{C}}} &= \phi_2 \eta_{\mathbf{B}} \end{aligned}$$

where $\eta_{\mathbf{A}}, \eta_{\mathbf{B}}$ and $\eta_{\mathbf{C}}$ represent the proportion of reactants $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ at a particular time t, and ϕ_1, ϕ_2 are the rate constants.

For our example, the initial conditions are: $\eta_{\mathbf{A}} = 1, \eta_{\mathbf{B}} = \eta_{\mathbf{C}} = 0$ at $t = 0$. Even though this linear model can be solved analytically, we will solve it numerically using the **lsode** solver. In this way, we want to imitate the difficulty of solving "real" Bayesian estimation problems in which the model evaluation step is expensive.

The goal of this example is to estimate the distribution function of parameters in the model, ϕ_1, ϕ_2, given the observations of $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and having a prior knowledge of those parameters in terms of their distribution function. Following the Box and Draper paper, we assign the prior distribution function of the logarithms of the rate constants, $\theta_i = \ln(\phi_i), i = 1, 2$, to be "non-informative" and locally uniform distributed. Furthermore, we also have two sets of independent multivariate observations, each at six different times. They are shown in table 3.1.

Time	y_{1u}	y_{2u}	y_{3u}
$\frac{1}{2}$	0.959	0.025	0.028
$\frac{1}{2}$	0.914	0.061	0.000
1	0.855	0.152	0.068
1	0.785	0.197	0.096
2	0.628	0.130	0.090
2	0.617	0.249	0.118
4	0.480	0.184	0.374
4	0.423	0.298	0.358
8	0.166	0.147	0.651
8	0.205	0.050	0.684
16	0.034	0.000	0.899
16	0.054	0.047	0.991

Table 3.1: Data for kinetic modeling example.

The relationship between observations and model responses can be written as follows:

$$\begin{aligned} y_{1u} &= \eta_{\mathbf{A}} + \epsilon_{1u} \\ y_{2u} &= \eta_{\mathbf{B}} + \epsilon_{2u} \\ y_{3u} &= \eta_{\mathbf{C}} + \epsilon_{3u} \end{aligned}$$

where u denotes the u-th observation, and ϵ_{iu} is a random error of observation. In this example, we have 12 observations, and we assume that the observations y_{1u}, y_{2u}, y_{3u} follow a multivariate normal distribution. From such an assumption, and the use of the "non-informative" and locally uniform prior distribution, equation (3.4) gives the posterior marginal density of parameters,

$$f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \;=\; C^{-1}\,|\mathbf{v}|^{-\frac{n}{2}},$$

where the element of matrix $\mathbf{v}$ is defined as

$$v_{i,j} \;=\; \sum_{u=1}^{n} \epsilon_{iu}\;\epsilon_{ju},$$

and n is the number of observations. C represents the normalizing factor which can be calculated as follows:

$$C \;=\; \int |\mathbf{v}|^{-\frac{n}{2}}\; d\boldsymbol{\theta}.$$

We can use Monte Carlo methods to calculate the normalizing factor C, and for this example, we use two different sources of sampling points: the standard **C** language random number generator, and the Hammersley-Wozniakowski sampling points described in chapter 2. We also apply the antithetic variates technique given in section 2.5.3 to reduce the variance of the normalizing factor estimator. Figure 3-2 shows the results.

As expected, the Hammersley-Wozniakowski method achieves a better convergence rate

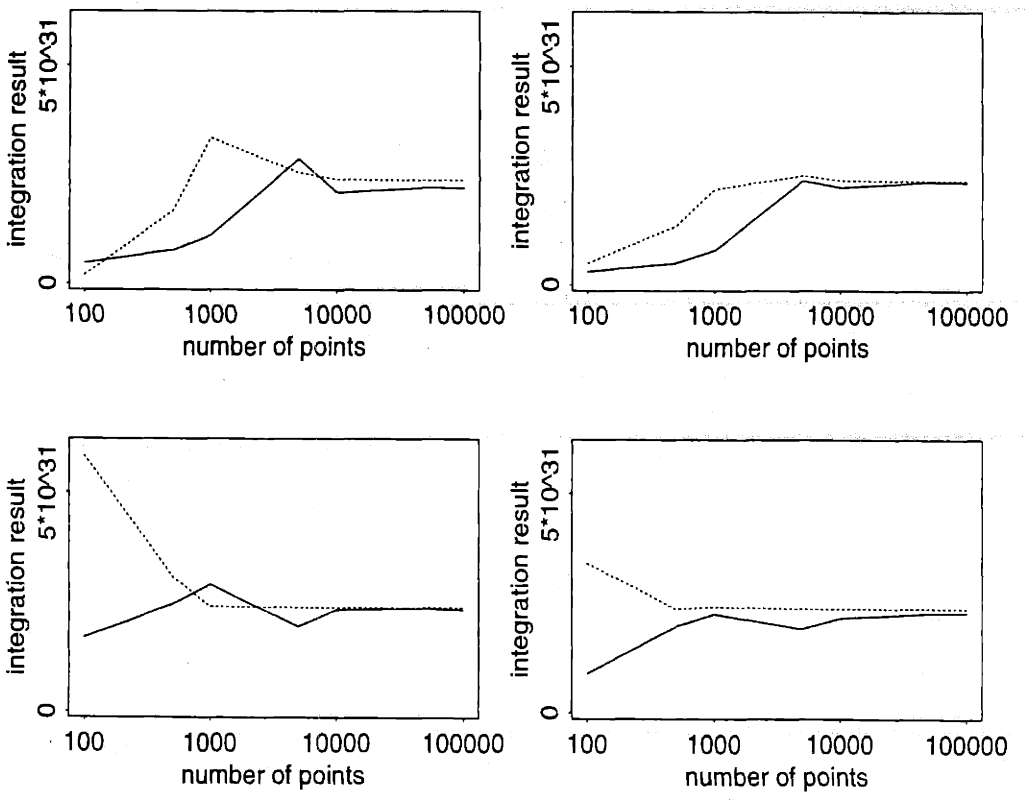


Figure 3-2: Normalization factor calculation as a function of the number of sampling points; upper figures are produced from the domain $[0,1] \times [0,1]$; lower figures are produced from the reduced domain $[0.1, 0.4] \times [0.1, 0.9]$; the right figures are produced with the antithetic variates technique; solid lines denote results from the standard C language sampling points, and dotted lines are for the results from the Hammersley-Wozniakowski method.

than the standard C language generator. Furthermore, the antithetic variates technique is able to reduce the fluctuations for both methods. Reducing the domain of integration to where the values of the posterior distribution are not negligible helps both methods as well. Various approaches for increasing such a convergence rate are needed in this example, because we have to solve the model at each sampling point. Consequently, a small increase in the convergence rate may save a lot of computing time. We see that the use of Hammersley-Wozniakowski sampling points along with the antithetic variates technique may reduce the cost of computation.

After the normalization factor is obtained, we can calculate the posterior marginal density for each parameter by marginally integrating again the normalized posterior joint marginal distribution. One of advantages for having the marginal density of a parameter is that it can provide not only the modal value of the parameter, but also the uncertainty description of that parameter, such as its dispersion and tail of distribution. Such measures as mentioned earlier are important inputs to the uncertainty analysis. Unfortunately, computing the posterior marginal density can be very expensive, therefore, we may prefer to generate their approximations. We use Gibbs sampling scheme to obtain an approximation to those marginal densities.

For this example, we implement the griddy-Gibbs sampling scheme to generate the marginal density function of parameters ϕ_1 and ϕ_2. We, first, produce 400 uniform sampling points inside the domain of parameters (we use Hammersley-Wozniakowski sampling points), then for each sampling point we generate a corresponding sampling point of the posterior distribution by applying the Gibbs sampling scheme with 30 steps. In each step, we need to take sampling points from a conditional distribution. Therefore, this conditional distribution is discretized into 20 equal-distance points, and the piece-wise constant distribution method is applied to produce sampling points.

The results can be seen in figures 3-3 to 3-5. From the marginal density at the 30-th iteration, we can easily estimate the modal position of the posterior distribution. Figure 3-5 shows that the posterior distribution is concentrated in a small region. It means that the error of parameters estimation from the use of those 12 observations is quite small. The results from applying Gibbs sampling scheme are close to the results shown in the Box and Draper paper [17].

For some applications, we may need the modal position of parameters. We can find the modal position directly by using an optimization method. Hence, the problem is formulated as follows:

$$\max_{\boldsymbol{\theta}} \; f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}).$$

Since the posterior distribution may have several modes, then the "simulated annealing" method as a global optimization method is suitable for this problem.

For our example, we choose the initial guess point and "temperature" are $(0.5, 0.5)$ and 1.0 respectively. The reduction of the "temperature" is scheduled according to the following equation,

$$T_{i+1} \;=\; 0.5\, T_i.$$

The initial point, "temperature", and such a schedule are appropriate for this simple example. The optimal solution for this example is 2.169×10^{34}, and located at the point $(0.20757, 0.49544)$. It should be noted that the value of the optimal solution comes from

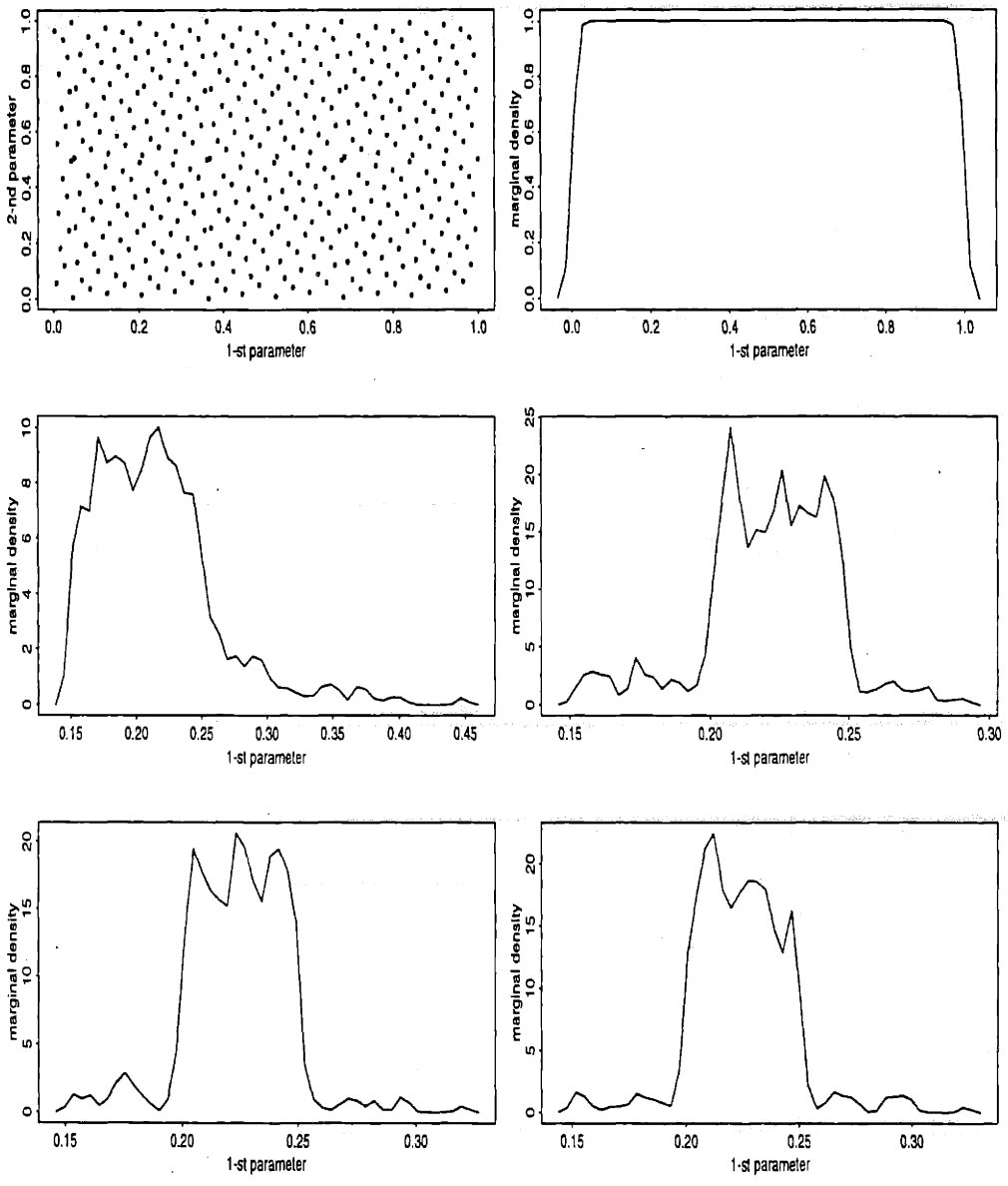


Figure 3-3: Scatter plot for two parameters; Evolution of marginal density for the first parameter at 0, 1, 10, 20, and 30 iterations of Gibbs sampling.

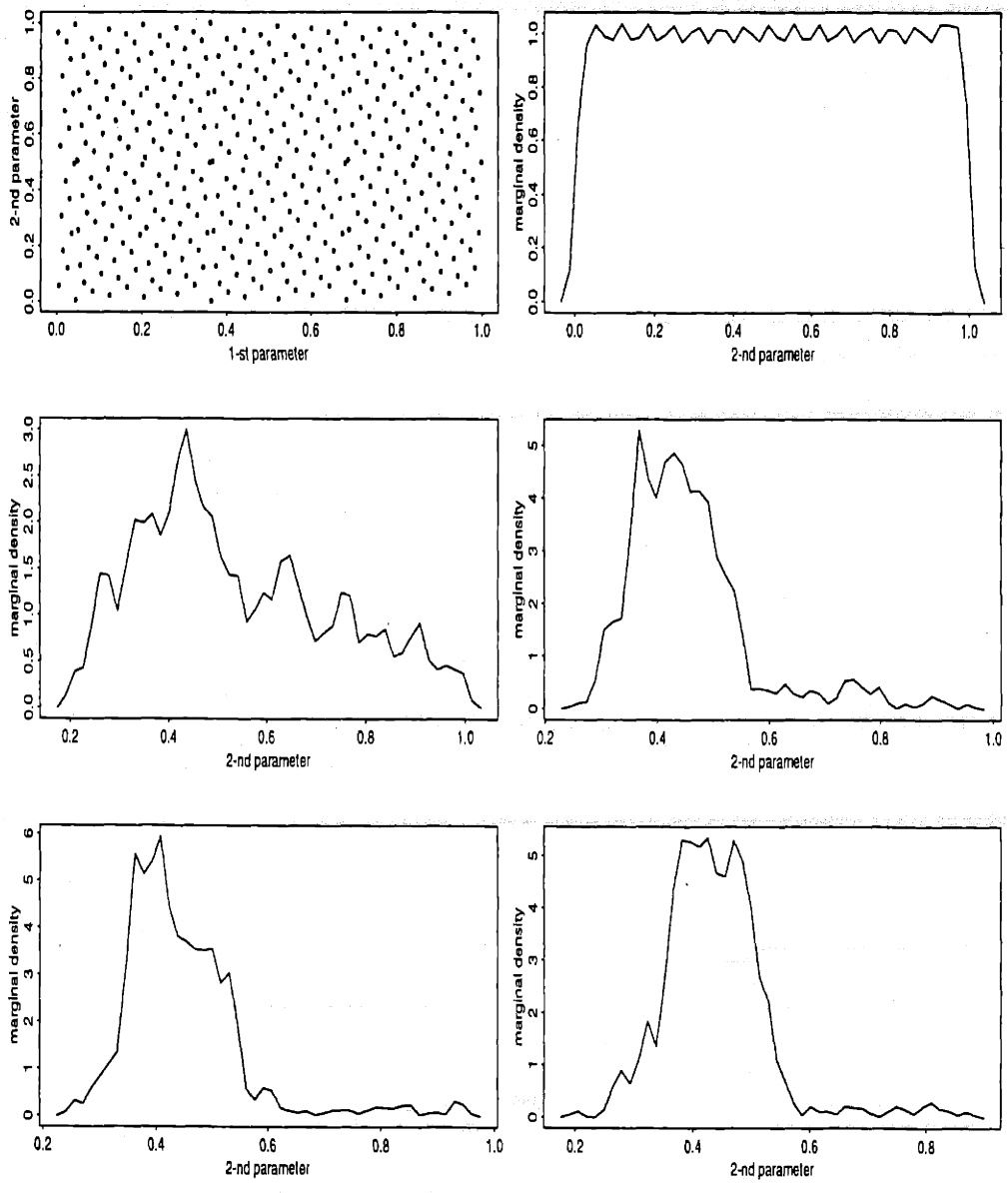


Figure 3-4: Scatter plot for two parameters; Evolution of marginal density for the second parameter at 0, 1, 10, 20, and 30 iterations of Gibbs sampling.

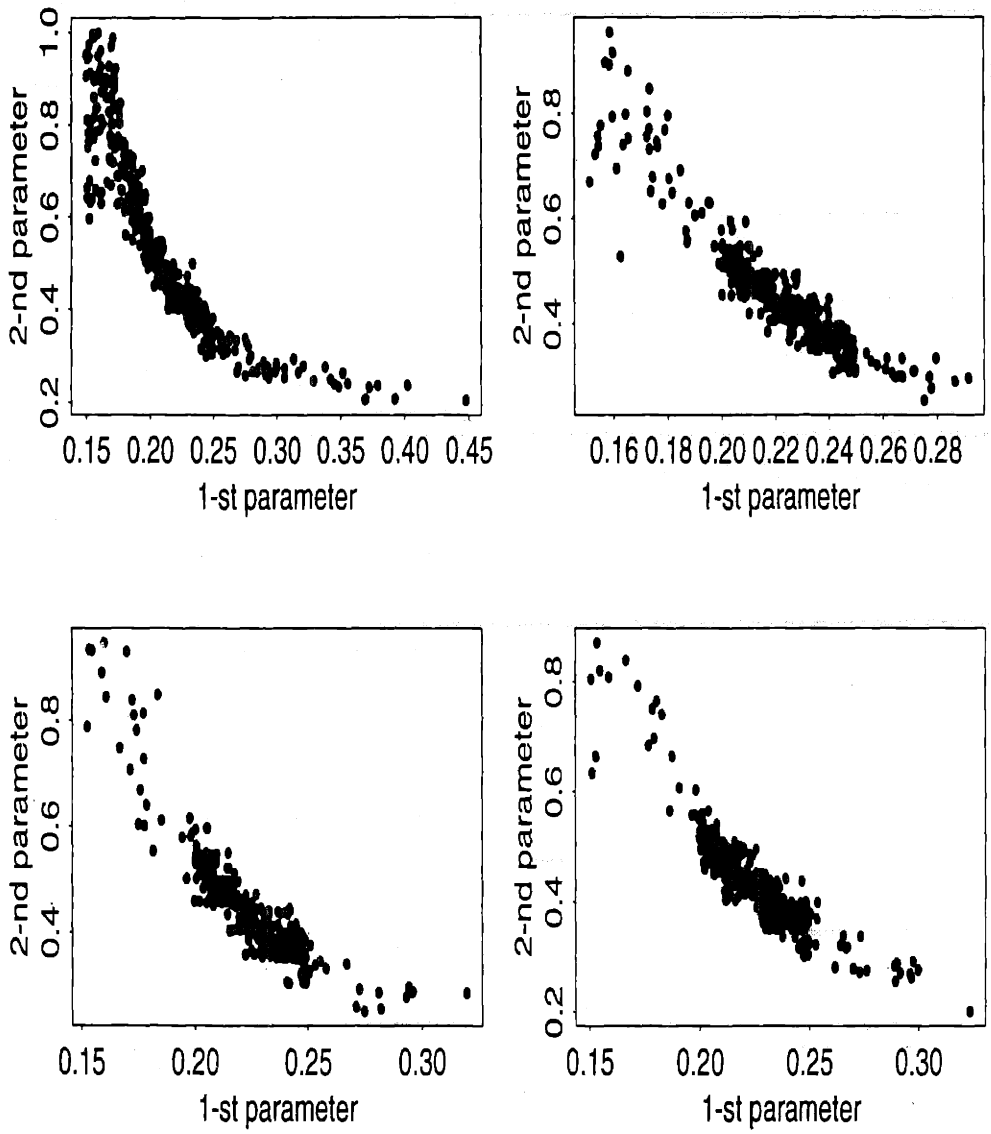


Figure 3-5: Evolution of sampling points for two parameters at 1, 10, 20, and 30 iterations of Gibbs sampling.

the unnormalized posterior distribution. After normalization, it becomes 911.345. The number of function evaluations is 1401, and the number of accepted evaluations is 387. These numbers show that the method is quite efficient. A complete description of the result can be seen in appendix A.

From the above discussions, we see that the application of the Bayesian approach requires a prior distribution function. It is clearly possible that different priors will yield different posteriors, which also means that different users may have different results. Such a possibility ignites the debate of subjectivism in the Bayesian framework: how does one know that one uses an accountable prior distribution. The answer to this question may come from the development of a methodology by which we can obtain "non-informative" prior distributions for performing a Bayesian inference. A well known method for generating such a prior is the maximum entropy method.

3.5.2 Maximum Entropy and Minimum Cross-Entropy Approaches

Before we discuss the maximum entropy and minimum cross-entropy approaches, we need to define the concept of entropy used in this context. The notion of entropy in information theory was introduced by Shannon [147] in developing a measure to assess the uncertainty of an outcome from a set of possible events whose probabilities of occurence are known. Such a measure should have the following properties:

1. The measure should be continuous in each probability of occurence.
2. If all the probabilities of occurence are equal, $\Pr(E_i) = \frac{1}{n}$, where E_i denotes the event i, and n is the number of events, then the measure should be a monotonic increasing function of n.
3. If an event can be broken down into two successive events, the original measure should be the weighted sum of the individual values of measures.

It can be shown that the only measure satisfying the three above requirements is of the form [147]:

$$H_S = -K \sum_{i=1}^{n} \Pr(E_i) \log(\Pr(E_i)),$$

where $S = \{E_i\}_{i=1}^n$ is the set of possible events, and H denotes the measure of uncertainty, information, and choice. Since such a form of H also arises in a certain formulation of statistical mechanics where $\Pr(E_i)$ is the probability of a system being in cell i of its phase space, then Shannon proposed to call $H = -\sum_{i=1}^n \Pr(E_i) \log(\Pr(E_i))$ as the entropy of the set of probabilities $\Pr(E_i); i = 1, \ldots, n$. Therefore, for a continuous real-valued random variable, x with probability density function $f_x(x)$, the entropy of this random variable can be written as follows:

$$H_x = -\int_{-\infty}^{\infty} f_x(x) \ln(f_x(x)) \, dx.$$

The entropy H_x is closely related to the disorder or uncertainty associated with making possible realizations or samples of the random variable x. As an example, only when we are certain of the outcome of x does H_x vanish, otherwise H_x is positive. It also means that we are more uncertain for the outcome of random variable x than that of random variable

y, if $H_x > H_y$. For this reason, maximizing the entropy while satisfying some constraints becomes a method for finding a probability density function that represents the highest uncertainty or, equivalently, a state of ignorance.

Jaynes [70] stated this argument formally, and it is known as the Jaynes' principle: The maximum entropy distribution is uniquely determined as the one which is maximally non-committal with regard to missing information, and that it agrees with what is known, but expresses maximum uncertainty with respect to all other matters. In other words, the least prejudiced or biased probability distribution is that which maximizes the entropy subject to a given information. A procedure for finding the maximal entropy density function which satisfies a set of moment constraints will be given in the next chapter.

The idea of maximum entropy method can be extended to include cases in which we have not only information about the constraints, but also a prior estimate of the density. What we have now is the principle of minimum cross-entropy which is known as a general method of inference for an unknown probability density when there exists a prior density and new information in the form of constraints on expected values [149]. This principle states that, of all the densities that satisfy the constraints, one should choose the posterior $q_x(x)$ with the least cross-entropy $H_{q,p} = \int q_x(x) \ln\left(\frac{q_x(x)}{p_x(x)}\right) dx$, where $p_x(x)$ is a prior estimate of $q_x(x)$. The principle of minimum cross entropy, actually, was first introduced with other names: the minimum directed divergence, or the minimum discrimination information [88].

Similar to the maximum entropy approach (see the next chapter), the minimum cross-entropy method will generate the least biased posterior distribution from a given prior and a set of expected constraints,

$$\begin{aligned} \min_{q_x(x)} \quad & \int q_x(x) \ln\left(\frac{q_x(x)}{p_x(x)}\right) dx \\ \text{s.t.} \quad & \int g_i(x)\, q_x(x)\, dx = \beta_i; \quad i = 1, \ldots, n, \end{aligned}$$

and the optimal solution will be of the form:

$$q_x(x) = p_x(x) \exp\left(-\sum_{i=1}^{n} \lambda_i\, g_i(x)\right),$$

where λ_i are Lagrange multipliers from the optimization result. The above optimization problem can be extended to include conditional densities which naturally appear in the updating scheme,

$$\begin{aligned} \min_{f_{\theta|y}(\boldsymbol{\theta}|\mathbf{y})} \quad & \int f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \ln\left(\frac{f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})}{f_\theta(\boldsymbol{\theta})}\right) d\boldsymbol{\theta} \\ \text{s.t.} \quad & \int g_i(\boldsymbol{\theta} \mid \mathbf{y})\, f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})\, d\boldsymbol{\theta} = \beta_i(\mathbf{y}); \quad i = 1, \ldots, n. \end{aligned}$$

For a specific value of $\mathbf{y}$, the latter optimization problem is in fact the same as the earlier one. Therefore, the optimal solution for the conditional optimization problem can be written as follows,

$$f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) = f_\theta(\boldsymbol{\theta}) \exp\left(-\sum_{i=1}^{n} \lambda_i(\mathbf{y})\, g_i(\boldsymbol{\theta} \mid \mathbf{y})\right).$$

If we choose $g_1(\boldsymbol{\theta} \mid \mathbf{y}) = 1$ and $g_2(\boldsymbol{\theta} \mid \mathbf{y}) = \ln\left(l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})\right)$, where $l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})$ is the likelihood function, then the optimal solution from minimizing the cross-entropy corresponds to the Bayesian updating scheme,

$$f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \;=\; f_\theta(\boldsymbol{\theta}) \, \exp\left(-\ln\left(f_y(\mathbf{y})\right) + \ln\left(l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})\right)\right), \tag{3.5}$$

because the likelihood function is derived from the conditional density function for $\mathbf{y}$, but by fixing $\mathbf{y}$ and varying $\boldsymbol{\theta}$,

$$l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \;=\; f_{y|\theta}(\mathbf{y} \mid \boldsymbol{\theta}),$$

and $f_y(\mathbf{y}) = \int f_{y|\theta}(\mathbf{y} \mid \boldsymbol{\theta}) \, d\boldsymbol{\theta}$. Clearly, equation (3.5) can be rewritten as

$$f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \;=\; \frac{l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \; f_\theta(\boldsymbol{\theta})}{f_y(\mathbf{y})}.$$

This result suggests that the posterior distribution based on the Bayesian updating scheme is the minimal cross-entropy distribution when the information of the input (the second constraint in the optimization) is of the form,

$$M_i \;=\; \int \ln\left(l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})\right) \; f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \, d\boldsymbol{\theta},$$

where M_i represents a measure for the information content of the likelihood function as an input to the updating scheme.

A similar result, but derived differently, was obtained by Zellner [199]. Zellner used the concept of the information processing rule (IPR) to produce a principle called the information conservation principle (ICP): Input information = Output information. This principle suggests the notion of efficiency for a certain IPR. For an inefficient IPR, its ouput information, measured in a suitable metric, is less than its input information. On the other hand, a good and efficient IPR will satisfy the ICP.

To implement this concept of efficiency, there is a need to measure the information in the input and output, in this case, the probability density functions. As an example, one can use postdata measures which determine the information in a probability density function as an average of its logarithmic with the posterior distribution as a "weight function". The inputs for the Bayesian approach are the likelihood function and the prior distribution function, and its outputs are the posterior distribution of parameters and the probability of responses. This input and ouput information will be of the form:

$$\begin{aligned}
M_l &= \int_{\boldsymbol{\theta}} f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \; \ln\left(l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})\right) \; d\boldsymbol{\theta} \\
M_a &= \int_{\boldsymbol{\theta}} f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \; \ln\left(f_\theta(\boldsymbol{\theta})\right) \; d\boldsymbol{\theta} \\
M_p &= \int_{\boldsymbol{\theta}} f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \; \ln\left(f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})\right) \; d\boldsymbol{\theta} \\
M_r &= \int_{\boldsymbol{\theta}} f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \; \ln\left(f_y(\mathbf{y})\right) \; d\boldsymbol{\theta},
\end{aligned}$$

where M_l and M_a are the input information, and M_p and M_r are the output information. It is clear that the information of input required in the minimum cross-entropy method has

the same form as these postdata measures. If we optimize the difference between output and input information, the optimal posterior distribution should be as follows [199],

$$f_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y}) \;=\; \frac{l_{\theta|y}(\boldsymbol{\theta} \mid \mathbf{y})\, f_{\theta}(\boldsymbol{\theta})}{f_{y}(\mathbf{y})}.$$

It means that the Bayesian updating scheme is an optimal processing rule with the above postdata measures.

Chapter 4

Random Variables Representation

One of the most critical elements required to directly and systematically treat any uncertain parameters or inputs in a model is the use of a correct representation for those uncertain parameters or inputs. Based on the arguments in chapter 3, we choose to use the probabilistic representation for uncertain parameters and variables in the model. It means that we have to study and develop the representations of random variables, random processes, and random fields.

This chapter focuses on the study of two complementary ways to represent any random variables; the indirect and direct methods [68]. Random processes and fields representations will be described in the next chapter. In this chapter, we will also study the application of a least-square technique for obtaining coefficients of polynomial chaos representations, and illustrate several methods for generating the probability density function of a random variable.

4.1 Indirect methods

A random variable can be distinguished or classified by its probability density function. For example, there are random variables of Gaussian distribution, of gamma distribution, or of Poisson distribution. Therefore, one way to represent a random variable indirectly is to use a general representation of a probability distribution, such as Edgeworth series expansion, or H-function [158]. In subsection 4.4, we will see how one can use a general parametric form of a probability density function to represent any probability density function.

These indirect methods are useful for models involving probability computations such as risk assessment [65], stochastic flexibility index [124, 163, 165], reliability test [169] and estimation [20, 58, 194], threshold analysis [172], chance-constrained and stochastic programming problems [188], and many others. In other words, these methods are needed when the mathematical conditions which define the problem apply to the distribution function.

Other indirect ways to represent a random variable are the quantile representation, such as the Cornish-Fisher expansion, and characterisitic function representation or other integral transforms which are suitable for analyzing distributions of sums, differences, products, quotients, and powers of continuous random variables. The Cornish-Fisher expansion approximates the quantiles of a distribution in terms of a polynomial function of unit Gaussian quantiles. The derivation of this expansion starts from having an identity between unit

Gaussian quantiles and the actual distribution quantiles,

$$\int_{-\infty}^{U_\alpha} Z(x)\,dx \;=\; \int_{-\infty}^{X_\alpha} g(x)\,dx,$$

where $Z(x)$ denotes the density of unit Gaussian, and $g(x)$ is the actual density function of interest. Expanding the left-hand side of the above equation into its Taylor series expansion at the point X_α, and expanding the right-hand side based on the property of cumulants of $Z(x)$ and $g(x)$ [72], we have the following identity,

$$\left[\sum_{j=1}^{\infty}\left\{\frac{(X_\alpha-U_\alpha)^j}{j!}\right\}H_j(X_\alpha)\right]Z(X_\alpha) \;=\; \left[\sum_{i=0}^{\infty} D^{-1}\,\frac{\left\{\sum_{j=1}^{\infty}\epsilon_j\frac{(-D)^j}{j!}\right\}^i}{i!}\right]Z(X_\alpha),$$

where $H_j(\cdot)$ is the j-th Hermite polynomial, and D is the differentiation operator. It is possible to rearrange the above equation to give two identities:

$$X(U_\alpha) \;=\; U_\alpha+\frac{1}{6}\left(U_\alpha^2-1\right)\kappa_3+\frac{1}{24}\left(U_\alpha^3-3U_\alpha\right)\kappa_4-\cdots, \tag{4.1}$$

$$U(X_\alpha) \;=\; X_\alpha-\frac{1}{6}\left(X_\alpha^2-1\right)\kappa_3-\frac{1}{24}\left(X_\alpha^3-3X_\alpha\right)\kappa_4+\cdots, \tag{4.2}$$

where κ_i represents the i-th cumulant. These cumulants can be calculated from the moments values, and vice versa [169],

$$\begin{aligned}\kappa_1 &= m_1\\ \kappa_{i+1} &= m_{i+1}-\sum_{j=1}^{i}\binom{i}{j}m_j\kappa_{i-j+1}\end{aligned}$$

and

$$\begin{aligned}m_1 &= \kappa_1\\ m_{i+1} &= \kappa_{i+1}+\sum_{j=1}^{i}\binom{i}{j}m_j\kappa_{i-j+1}.\end{aligned}$$

Johnson and Kotz [72] gave the relationships (4.1 – 4.2) for up to twelve terms of expansion. These two formulas assume that the actual density function of interest has been standardized such that it has zero mean value and unit variance. Therefore, if we want to calculate the quantile $X_{\alpha=0.5}$ for the density function $g(x)$, we first need to calculate or look up the value of $U_{\alpha=0.5}$ from the Gaussian table, and then we can use equation (4.1) along with the knowledge of cumulants of $g(x)$ to calculate that quantile, $X_{\alpha=0.5}$. To compute such a high order Cornish-Fisher expansion, one can use an algorithm, or the corresponding computer code, proposed by Lee and Lin [93].

As mentioned above, characteristic functions or other integral transforms provide the means for deriving and analyzing the distribution of algebraic functions of continuous random variables. Before we study the use of integral transforms to aid the generation of a distribution function, we need to define them properly. In general, the integral transform

is defined by [97]

$$\int_{-\infty}^{\infty} K(t,x)\, dF(x),$$

where $F(x)$ is a distribution of function, and $K(t,x)$ is a suitable kernel which is a function of the random variable x and contains also a parameter t which could be either discrete or continuous.

We can now list a few possibilities for the choice of $K(t,x)$ which are important in the study of distribution functions [97, 158].

$$\begin{array}{lll}
(A) & K(t,x) = x^t & \\
(B) & K(t,x) = |x|^t & \\
(C) & K(t,x) = x^{(t)} = x(x-1)\cdots(x-t+1), & x^{(0)} = 1 \\
(D) & K(t,x) = e^{tx} & \\
(E) & K(t,x) = t^x & \\
(F) & K(t,x) = e^{itx} & \\
(G) & K(t,x) = e^{-tx};\ x \geq 0 & \\
(H) & K(t,x) = x^{t-1};\ x \geq 0 &
\end{array}$$

For the first three types of $K(t,x)$, the parameter t is restricted to non-negative integers, and for the rest, t is a continuous real-valued parameter. As a result, the transforms A, B, and C transform the distribution function $F(x)$ into sequences. Clearly from kernel A, the sequence of moments values is a product of an integral transform. Integral transforms with kernels D, E, F, G, and H are also known as the moment generating function, the probability generating function, the Fourier transform or the characteristic function, the Laplace transform, and the Mellin transform respectively. There are other integral and discrete transforms, which for many cases may also be valuable. For example, the generating function, the Z, the ζ, and the Walsh-Hadamard transforms for discrete random variables, or the Hankel transform for continous random variables with spherical distribution [158].

We can use these integral transforms for generating the density function of algebraic functions of random variables. As an example, we can use the Laplace transform to derive the distribution of sums of non-negative random variables. On the other hand, if the random variables may have both positive and negative values, the Fourier or bilateral Laplace transforms become the tools to derive the density function of their sums and differences. Similarly, the Mellin transform is a suitable apparatus for generating the density function of products and quotients of non-negative random variables, and for entire real line supported random variables, one can apply the modified Mellin transform. Examples for the applications of these integral transforms can be found elsewhere [126, 158, 173]. It should be noted that all random variables in those algebraic functions should be independent in order to make the integral transforms work.

Based on the property of Mellin transform, one can use a quite general function with many parameters to represent many common distributions. This function itself is a transcendental function introduced by Fox in 1961 [49], and called the H-function. As an example, H-function distribution can cover gamma, Weibull, beta, exponential and other type of distributions [158]. Furthermore, the probability density function of the product, power, or quotient of H-function variables is an H-function variable. It also means that the H-function can serve as a basis for handling algebraic functions of mixtures of such random variables. The form of the H-function is given in equation (4.3) which also defines

the Mellin-Barnes contour integral.

$$\mathbf{H}_{p,q}^{m,n}\left[z\left|\begin{array}{c}(a_1,\alpha_1),\ldots,(a_p,\alpha_p)\\(b_1,\beta_1),\ldots,(b_q,\beta_q)\end{array}\right.\right] = H(z)$$

$$= \frac{1}{2\pi i}\int_C \frac{\prod_{j=1}^{m}\Gamma(b_j-\beta_j s)\prod_{j=1}^{n}\Gamma(1-a_j+\alpha_j s)}{\prod_{j=m+1}^{q}\Gamma(1-b_j+\beta_j s)\prod_{j=n+1}^{p}\Gamma(a_j-\alpha_j s)}\, z^s ds \qquad (4.3)$$

where

$$\begin{array}{rcll} 0 \leq & m & \leq q, & \\ 0 \leq & n & \leq p, & \\ \alpha_j & > & 0 & \text{for } j = 1,2,\ldots,p, \\ \beta_j & > & 0 & \text{for } j = 1,2,\ldots,q, \end{array}$$

and a_j, b_j are complex numbers such that no pole of $\Gamma(b_j-\beta_j s)$ for $j=1,2,\ldots,m$ coincides with any pole of $\Gamma(1-a_j+\alpha_j s)$ for $j=1,2,\ldots,n$.

As an example, the gamma distribution can be rewritten in terms of the H-function distribution as follows:

$$\begin{aligned} f(x) &= \frac{x^{\theta-1}\exp\left(-\frac{x}{\phi}\right)}{\phi^\theta\,\Gamma(\theta)}, \quad x > 0; \quad \theta,\phi > 0 \\ &= \frac{1}{\phi\,\Gamma(\theta)}\mathbf{H}_{0,1}^{1,0}\left[\frac{1}{\phi}\,x\,|\,(\phi-1,1)\right] \end{aligned} \qquad (4.4)$$

where ϕ, and θ denote the scale and shape parameters respectively. Every type of distribution, which can be represented by H-function, has a similar form, *i.e.* a constant multiplied by a H-function.

$$f(x) = \begin{cases} k\,\mathbf{H}_{p,q}^{m,n}\left[c\,x\left|\begin{array}{c}(a_j,\alpha_j),\ldots,(a_p,\alpha_p)\\(b_j,\beta_j),\ldots,(b_q,\beta_q)\end{array}\right.\right], & x > 0 \\ 0, \quad \text{otherwise} \end{cases}$$

Since H-function approach can be used for generating the distribution of an algebraic function of independent random variables, this approach is suitable for empirical equations involving products, quotients, and powers. We will consider below a simple example of generating the distribution of Nusselt numbers from an empirical relationship between Nusselt and Graetz numbers.

Example 4.1.1

A simple empirical relationship between Nusselt and Graetz numbers for laminar-flow heat transfer can be written as [105]:

$$N_{Nu} = 2\,N_{Gz}^{\frac{1}{3}}\,\phi_v.$$

One may assume in the lack of knowledge that the Graetz number N_{Gz} and the correction factor ϕ_v to be of gamma distribution. In this case, let us consider that the mean value of N_{Gz} is 8.0, and its standard deviation equals to 2.0. Similarly for ϕ_v, its mean value and

standard deviation are 1.0, and 0.5 respectively. From the relationship between the shape and scale factors with the mean value and standard deviation of a gamma distribution, the probability density function for N_{Gz} and ϕ_v can be written in terms of the H-function (see equation (4.4)),

$$\begin{aligned} f_{N_{Gz}} &= \frac{1}{0.5\,\Gamma(16.0)}\,\mathbf{H}_{0,1}^{1,0}\left[\frac{1}{0.5}\,N_{Gz}\,|\,(0.5-1,1)\right] \\ f_{\phi_v} &= \frac{1}{0.25\,\Gamma(4.0)}\,\mathbf{H}_{0,1}^{1,0}\left[\frac{1}{0.25}\,\phi_v\,|\,(0.25-1,1)\right] \end{aligned}$$

To calculate the H-function distribution of Nusselt number N_{Nu}, we will use **STOFAN** program developed by Carter [27], and then we will use **TEST** subroutine from Hill [66] to approximate that distribution using the Laguerre series expansion. Input and output subroutines of these programs have been modified in order to be able to automatically generate an output file in LaTeX format, and to include the plots of density and cumulative functions (see appendix D for the output file of this example). As a result, the H-function distribution of N_{Nu} becomes

$$f_{N_{Nu}} = 3.2116\;10^{-13}\;\mathbf{H}_{0,2}^{2,0}\left[2.51984\,N_{Nu}\,|\,(3,1)\,(15.6667,0.3333)\right],$$

and the j-th moment of N_{Nu} can be computed from the following equation [158],

$$\mu_j = \frac{3.2116\;10^{-13}}{2.51984^{j+1}}\,\Gamma(4+j)\,\Gamma(16+0.3333\,j).$$

We can check this result by calculating the zeroth moment. From the above equation, we obtain $\mu_0 = 1.0$. Similarly, the mean value and standard deviation of N_{Nu} can be calculated, and they equal to 3.97186 and 4.45628 respectively.

For this example, the **TEST** program uses the first three moments to approximate the distribution using the Laguerre series expansion. The result has the following form,

$$\begin{aligned} \hat{f}_{N_{Nu}} = {} & N_{Nu}\,\exp\left(-N_{Nu}\right)\,\Big(1+0.9859\,(N_{Nu}-2)+0.1670\,\left(N_{Nu}^2-6N_{Nu}+6\right)+ \\ & 0.0016\,\left(N_{Nu}^3-12N_{Nu}^2+36N_{Nu}-24\right)\Big). \end{aligned}$$

The **TEST** program also gives the Pearson approximation of this distribution. The result is a type IV distribution,

$$\hat{f}_{N_{Nu}} = y_0\,\left(1+\frac{N_{Nu}^2}{4.28874^2}\right)^{-7.2347}\,\exp\left(15.4306\arctan\left(\frac{N_{Nu}}{4.28874}\right)\right),$$

where y_0 is the normalizing constant and its value is 5459.82. It should be pointed out that the calculation of parameter A for type IV of Pearson curves system with **TEST** is not correct. Therefore, we refer to the original equations [44] for calculating parameters in the Pearson curves system. The results for these approximations are given in figure 4-1. The detailed description on how to apply Pearson curves system and orthogonal series expansions to the approximation of a probability density function will be described in subsection 4.4.2.

Figure 4-1 shows that those two approximation schemes give different modal positions, implying that the maximum likelihood of the values of N_{Nu} will be different depending on

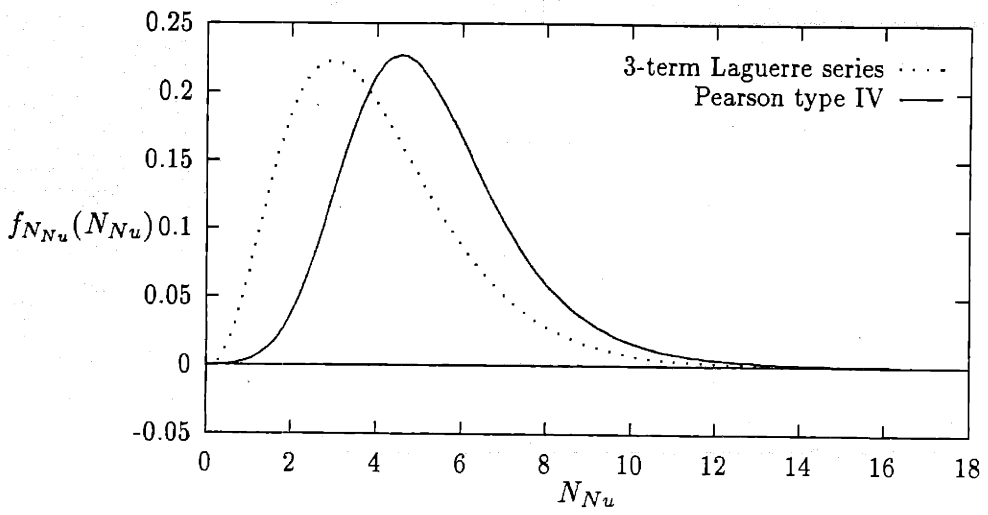


Figure 4-1: Approximation of Nusselt number density function by using a three-term Laguerre series expansion and by using a type IV of Pearson curves system.

the approximation scheme we choose. Nevertheless they produce the same trend and shape of the probability density function of N_{Nu}. From the Laguerre series expansion result (see appendix D), one can see that the 90 % confidence interval of N_{Nu} is around 1.36 – 7.76. On the other hand, if we use the mean values of the Graetz number and the correction factor, the Nusselt number value is 4.0. This suggests that our limited knowledge or data for those two variables can only give us such a range of values with 90 % confidence level. Furthermore, the maximum likelihood of N_{Nu} is around 2.96 which is quite different from the result of mean value analysis, that is 4.0, even though the mean value of N_{Nu} is 3.97. Therefore, if one uses the mean value as part of the design parameters, there is almost no safety factor to be used since $4.0 \approx 3.97$. On the contrary, if one uses the maximum likelihood value, then this safety factor should now be considered since it is around $\frac{4.0}{2.96} \approx 1.35$.

This H-function approach can be extended to problems with more than one dimension. The definition of H-function for two variables is given in Srivastava, Gupta, and Goyal's book [159]. The book also has a list of the H-function expressions for many elementary special functions. Such a list is useful for defining the form of the H-function for certain density functions. For further applications of H-function in statistics, one can refer to Mathai and Saxena [104].

As mentioned earlier, another indirect method to represent random variables is the use of orthogonal series expansions. The orthogonal series expansions have been used to assess the population of Grey Whales [22], to determine the probability of detecting a signal with an arbitrary signal-to-noise ratio fluctuation law [172], to predict the responses of an offshore structure which has substantial non-linear characteristics in random seas [117], to simulate a power system [169], to solve Bayesian inference problems with a non-normality assumption [54], to approximate the distribution of temperature in turbulent jets [35], to transform a probabilistic inequality constraint into its deterministic equivalent [166], and many others. A detailed discussion of these orthogonal expansions will be given in 4.4.2.

4.2 Direct methods

While indirect methods are applicable when the mathematical conditions or models, which define the problem, apply to the distribution function, direct methods are useful for mathematical models involving distributed variables. As an example, the stochastic differential and/or algebraic equations which contain random fields and/or variables are one of major area of applications of the direct method.

In 1963, Imamura, Meecham, and Siegel [68] proposed the use of an expansion of the random function from which the models, expressed by the differential equations, are transformed into models of the "coefficients" of the functional expansion. The functional expansion that they used is called Wiener-Hermite functionals which were introduced by Wiener [190] in 1938. This functional expansion can be considered as a generalization of a concept proposed in a series of papers by Edgeworth [43] in 1914 and extended independently by Baker [8] in 1934.

Edgeworth proposed that a wide class of probability density functions can be approximated sufficiently by "translating" a Gaussian density function according to the following operator relationship,

$$x \;=\; \xi \times a\,(1 + \kappa\xi + \lambda\xi^2),$$

where x is a random variable with density function of interest, and ξ is a standardized Gaussian distributed random variable. By this operation each little column with indefinitely small base $\Delta\xi$ is shifted or "translated" to a new distance from the median with a new base Δx, and this new base may be written as

$$\Delta x \;=\; \frac{dx}{d\xi}\,\Delta\xi \;=\; a'\,(1 + \kappa'\xi + \lambda'\xi^2)\,\Delta\xi.$$

Similar to this idea, Baker proposed the following procedure to generate a function, φ, such that a non-Gaussian distribution, $f_x(x)\,dx$, transformed by the transformation, $x = \varphi(\xi)$, becomes a standardized Gaussian distribution,

$$f_x\,(\varphi(\xi))\;\varphi'(\xi)\,d\xi \;=\; \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\xi^2}{2}\right)\,d\xi.$$

One can also regard the above equation as an identity in ξ,

$$f_x\,(\varphi(\xi))\;\varphi'(\xi) \;=\; \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\xi^2}{2}\right). \tag{4.5}$$

If it is assumed that $f_x(\cdot)$ and $\varphi(\cdot)$ are functions that can be represented by Maclaurin expansions, the left and right hand sides of equation (4.5) can be expanded, and the coefficients of corresponding powers of ξ can be equated in order to determine φ. Therefore, suppose that

$$f_x(t) \;=\; \sum_{i=0}^{\infty} A_i\,t^i$$

$$\varphi(t) \;=\; \sum_{i=0}^{\infty} B_i\,t^i$$

then one can compute the coefficients $\{A_i\}$ since $f_x(\cdot)$ is given, and the coefficients $\{B_i\}$ are obtained from expanding the left and right hand sides of equation (4.5). As a result, the coefficients $\{B_i\}$ will be functions of $\{A_i\}$. The transformation function then can be written as follows:

$$x = \sum_{i=0}^{\infty} B_i \, \xi^i.$$

Baker gave an example for approximating a density function,

$$f_x(x) = \frac{1}{\sqrt{2\pi}} 0.9929 \left(1 + \frac{1}{10}\right)^{99} \exp(-10x),$$

which is skewed in the positive direction. Using the formula given in his paper [8], the coefficients $\{B_i\}$ then become

$$\{B_1, B_2, B_3, B_4, B_5, \ldots\} = \{1.0072, 0.0511, 0.0050, -0.0080, 0.0004, \ldots\},$$

which means that the random variable x can be written as

$$x = 1.0072\xi + 0.0511\xi^2 + 0.0050\xi^3 - 0.0080\xi^4 + 0.0004\xi^5 + \cdots.$$

The probability density functions of $\hat{x} = 1.0072\xi + 0.0511\xi^2 + 0.0050\xi^3 - 0.0080\xi^4$ and of x are shown in figure 4-2. The density function of $\hat{x}$ is obtained by using the subroutine

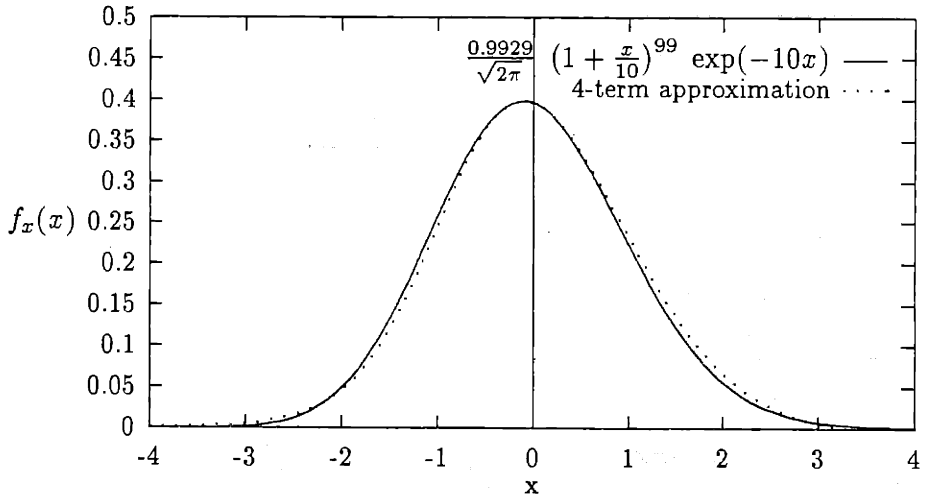


Figure 4-2: Comparison of the density functions of x and of its approximation, $\hat{x}$, from Baker approach.

CalDF given in appendix C. It is clear that for this example, we can closely approximate x as a function of the Gaussian random variable ξ.

Wiener introduced a similar approximation scheme for a random process [190], however,

he derived it by considering theory and applications of statistical mechanics. Based on the ergodicity property of a random process, his approach is implicity applicable to a random variable with a distribution similar to that of a random process. He called this type of random process as a type of chaos. This name is somewhat awkward for today's definition of chaos. Nevertheless, we will continue to use his definition, and clarify the term whenever it is not obvious.

He argued that since of all the forms of chaos occuring in physics, there is only one class, that is the Brownian motion, which has been studied with anything approaching completeness, then one can show that this type of chaos is central, and it allows one to approximate all types of chaos. As the Brownian motion, or Gaussian distribution, is the main ingredient for representing an arbitrary random variable, the concept of homogeneous and polynomial chaoses becomes its template.

The definitions of homogeneous and polynomial chaoses can be derived from the integral representation of chaos, however, a simpler definition, given by Ghanem and Spanos [56], is cited here.

Definition 4.1 *Let* $\{\xi_i(\omega), i = 1, \ldots, \infty\}$ *be a set of orthonormal Gaussian random variables. Consider the space* $\mathrm{S}(\Xi_p)$ *of all polynomials in* $\{\xi_i(\omega), i = 1, \ldots, \infty\}$ *of degree not exceeding p. Let* Ξ_p *represents the set of all polynomials in* $\mathrm{S}(\Xi_p)$ *orthogonal to* $\mathrm{S}(\Xi_{p-1})$. *Finally let* $\mathrm{SS}(\Xi_p)$ *be the space spanned by* Ξ_p. *Then, the subspace* $\mathrm{SS}(\Xi_p)$ *of* Θ *is called the p-th homogeneous chaos, and* Ξ_p *is called the polynomial chaos of order p, where* Θ *is a Hilbert space of functions defined by mapping the probability space,* Ω, *onto the real line,* $\mathbf{R}$.

Based on the argument of Wiener mentioned above and some properties of polynomial chaos [75], one can show that any square-integrable random variable can be approximated as closely as desired by a polynomial chaos expansion. Therefore, any random variable $x(\omega)$ from the space Θ admits the following representation,

$$\begin{aligned} \mathbf{x}(\omega) \;=\; & a_0\Xi_0 + \sum_{i_1=1}^{\infty} a_{i_1}\Xi_1(\xi_{i_1}(\omega)) + \sum_{i_1=1}^{\infty}\sum_{i_2=1}^{i_1} a_{i_1 i_2}\Xi_2(\xi_{i_1}(\omega),\xi_{i_2}(\omega)) \\ & + \sum_{i_1=1}^{\infty}\sum_{i_2=1}^{i_1}\sum_{i_3=1}^{i_2} a_{i_1 i_2 i_3}\Xi_3(\xi_{i_1}(\omega),\xi_{i_2}(\omega),\xi_{i_3}(\omega)) + \cdots \end{aligned} \tag{4.6}$$

As we can see now, the polynomial chaos expansion is a type of functional, since it involves random variables. It is a function that maps the probability space, Ω, onto the real line, $\mathbf{R}$. From this point of view, one can make a connection between this type of functional expansion with the functional Taylor series expansion [119]. One major difference is the use of orthonormal polynomial functionals in the former.

Since the elements of basis functions in the polynomial chaos expansion are Gaussian distributed random variables, the natural basis functions or orthonormal polynomials for a Gaussian weighting function are Hermite polynomials. Ghanem and Spanos showed a procedure to generate multi-dimensional basis functions for the polynomial chaos expansion [56]. A similar procedure also exists in the quantum field theory, and the corresponding basis functions are referred to Wick polynomials [100]. Based on the completeness of these Wick or Hermite polynomials in the $L_2(C)$ space, Cameron and Martin showed that the polynomial chaos expansion is convergent in the mean-square sense [24].

To clarify the notion of polynomial chaos expansions, we will consider two simple ex-

amples of approximating log-normal and exponential distributed random variables. The problem is to find the coefficients in the polynomial chaos expansion which satisfy the least-square error criterion (see the next section).

Example 4.2.1

A random variable, x, has the following density function

$$f_x(x) = \frac{1}{0.5x\sqrt{2\pi}} \exp\left(-\frac{(\ln(x)-1.0)^2}{2\ 0.5^2}\right); \quad 0 \le x \le \infty.$$

We will use the knowledge of moments values to generate its polynomial chaos expansion. From that probability density function, we can calculate its mean value, μ, variance, σ_2, skewness, γ_1, and kurtosis, γ_2, which are given below.

$$\begin{aligned} \mu &= 3.08022 \\ \sigma^2 &= 2.69476 \\ \gamma_1 &= 1.75018 \\ \gamma_2 &= 8.89841 \end{aligned}$$

Since we have four moments values, we can generate the corresponding polynomial chaos expansion for up to four terms,

$$x = x_0 + x_1\xi + x_2(\xi^2 - 1) + x_3(\xi^3 - 3\xi),$$

and its mean value, variance, skewness, and kurtosis can be written as follows.

$$\begin{aligned} \mu_A &= x_0 \\ \sigma_A^2 &= x_1^2 + 2x_2^2 + 6x_3^2 \\ \mu_{3A} &= 8x_2^3 + 108x_2x_3^2 + 36x_1x_2x_3 + 6x_1^2x_2 \\ \mu_{4A} &= 60x_2^4 + 2232x_2^2x_3^2 + 576x_1x_2^2x_3 + 60x_1^2x_2^2 + 3348x_3^4 + \\ &\quad 1296x_1x_3^3 + 252x_1^2x_3^2 + 24x_1^3x_3 + 3x_1^4 \\ \gamma_{1A} &= \frac{\mu_{3A}}{\sigma_A^3} \\ \gamma_{2A} &= \frac{\mu_{4A}}{\sigma_A^4} \end{aligned}$$

The generation of the moments values for a polynomial chaos expansion uses the following formula for Gaussian random variables [120],

$$E(\xi^n) = \begin{cases} 0 & n = 2k+1 \\ 1 \cdot 3 \cdots (n-1) & n = 2k, \end{cases} \tag{4.7}$$

as an example, the variance, σ_A^2, is obtained from conducting the following steps,

- According to the definition of variance

$$\sigma_A^2 = E\left(\left(x_1\xi + x_2(\xi^2 - 1) + x_3(\xi^3 - 3\xi)\right)^2\right)$$

- From orthonormal property of Hermite polynomials in the polynomial chaos expan-

sion, the cross-products become zero.

$$\sigma_A^2 = x_1^2 E\left(\xi^2\right) + x_2^2 E\left((\xi^2-1)^2\right) + x_3^2 E\left((\xi^3-3\xi)^2\right)$$

- Expanding all terms inside each expectation operator, and invoking the linearity property of expectation operation, we have

$$\sigma_A^2 = x_1^2 E\left(\xi^2\right) + x_2^2\left\{E\left(\xi^4\right) - 2E\left(\xi^2\right) + 1\right\} + x_3^2\left\{E\left(\xi^6\right) - 6E\left(\xi^4\right) + 9E\left(\xi^2\right)\right\}$$

- Using equation (4.7), we can evaluate all expectation operation terms

$$\sigma_A^2 = x_1^2 + 2x_2^2 + 6x_3^2$$

From the knowledge of moments values, we can formulate the least-square criterion to compute the coefficients in the expansion.

$$\begin{aligned} \min \quad & \sum_{i=1}^{3} f_i^2 \\ \text{s.t.} \quad & \mu_A - \mu = 0 \\ & \sigma^2 - \sigma_A^2 = f_1 \\ & \gamma_1 - \gamma_{1A} = f_2 \\ & \gamma_1 - \gamma_{1A} = f_3. \end{aligned}$$

Solving the above formulation using **GAMS** with **MINOS** solver [21] yields the following optimal coefficients,

$$\begin{aligned} x_0 &= 3.08022 \\ x_1 &= 1.54271 \\ x_2 &= 0.36472 \\ x_3 &= 0.09014. \end{aligned}$$

Figure 4-3 shows the true density function of x and the density functions of its polynomial chaos expansions. The density functions for 3 and 4 terms of expansions are calculated using the **CalDF** subroutine given in appendix C.

Example 4.2.2

An exponentially distributed random variable, y, is considered for our second example. Its probability density function can be written as follows:

$$f_y(y) = \exp(-y); \quad 0 \le y \le \infty.$$

Using the same approach described in the previous example, we generate 2, 3, and 4 term of polynomial chaos expansions. Figure 4-4 shows the accuracy of polynomial chaos expansion results in terms of the density function of y.

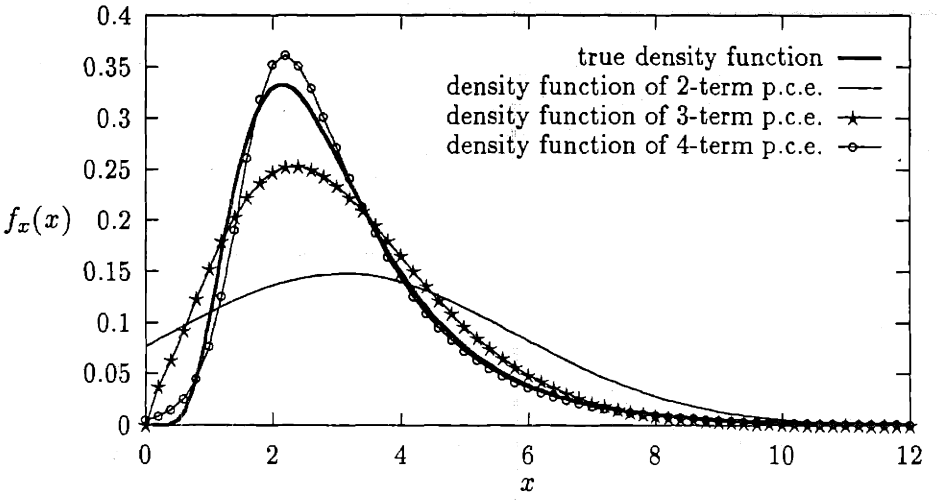


Figure 4-3: Comparison of the density functions of x and of its approximations by polynomial chaos expansion approach.

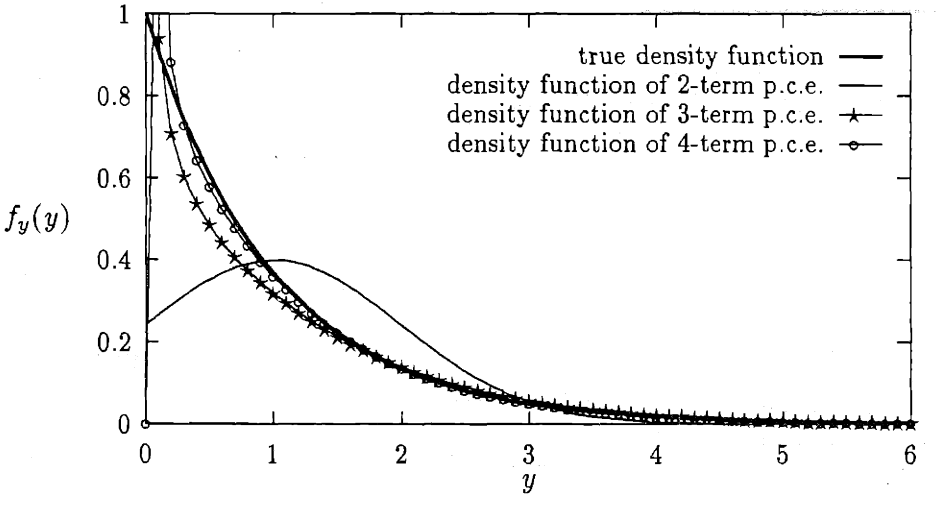


Figure 4-4: Comparison of the density functions of y and of its approximations by polynomial chaos expansion approach.

4.3 Least-Square technique

An optimization scheme can be applied to a random variable with a set of finite moments to generate its polynomial chaos expansion. The least sum of square differences between the true moments and their approximations gives an optimal set of coefficients in the expansion. It means that the moments of expansion may not match exactly with the true moments, however, the use of such an optimization scheme makes all coefficients in the expansion to be of real value. As an example, a log-normal random variable, a, with distribution

$$f_a(a) = \frac{1}{a\sqrt{2\pi}} e^{-\frac{\ln(a)^2}{2}} ; \quad 0 \le a \le \infty$$

cannot have a three-term polynomial chaos expansion that has mean value, variance, and coefficient of sknewness exactly the same as its true values. For this example, the true mean value, variance, and coefficient of sknewness are given respectively as follows:

$$\begin{aligned} \mu_T &= e^{\frac{1}{2}} \\ \sigma_T^2 &= e\,(e-1) \\ \gamma_{1T} &= (e-1)^{\frac{1}{2}}\,(e+2). \end{aligned} \tag{4.8}$$

Similarly the mean value, variance, and coefficient of sknewness of the expansion are defined below:

$$\begin{aligned} \mu_A &= a_0 \\ \sigma_A^2 &= a_1^2 + 2a_2^2 \\ \gamma_{1A} &= \frac{6a_1^2a_2 + 8a_2^3}{\sigma^3}, \end{aligned} \tag{4.9}$$

where a_0, a_1, and a_2 are the coefficients of polynomial chaos expansion. Equating equations (4.8) with (4.9), and then solving them always produces at least one complex valued coefficient. On the other hand, the use of least-square technique to this example,

$$\begin{aligned} \min \quad & \sum_{i=1}^{3} f_i^2 \\ \text{s.t.} \quad & \mu_T - \mu_A = f_1 \\ & \sigma_T^2 - \sigma_A^2 = f_2 \\ & \gamma_{1T} - \gamma_{1A} = f_3, \end{aligned}$$

yields only real coefficients because the true moments may not be matched exactly. In general, for a random variable with M number of independent information, such as its moments, skewness, or other measures of distribution, the formulation becomes

$$\begin{aligned} \min \quad & \sum_{i=1}^{M} f_i^2 \qquad\qquad\qquad\qquad (\text{P1}) \\ \text{s.t.} \quad & m_{iT} - m_{iA} = f_i \; ; \; i = 1, \ldots, M \end{aligned}$$

where m_i is an independent measure of probability distribution.

Other constraints which will improve the approximation can be added to this formula-

tion. As an example, if one wants to calculate coefficients of polynomial chaos expansion for a log-normal random variable, a constraint that makes such a polynomial chaos expansion to have a continuous distribution from 0 to ∞ should be included in the formulation (P1). In order to develop this additional constraint, one needs to consider a technique for generating the probability distribution of a random variable which is a function of a Gaussian random variable. The following fundamental theorem [120] provides a mechanism to generate that distribution.

Theorem 4.1 *Suppose that $y = g(x)$, and x is a random variable with probability density function $f_x(x)$, then to find the probability density function of y, $f_y(y)$, for a specific y, the equation $y = g(x)$ should be solved. Denoting its real roots by x_n,*

$$y = g(x_1) = \cdots = g(x_n) = \cdots$$

one can show that

$$f_y(y) = \frac{f_x(x_1)}{|g'(x_1)|} + \cdots + \frac{f_x(x_n)}{|g'(x_n)|} + \cdots$$

where $g'(x)$ is the derivative of $g(x)$.

As stated in the above theorem, in order to have a continuous distribution from $y = L$ to $y = U$, equation $y = g(x)$ must have at least one real root for $y \in [L, U]$. Since the form of $g(x)$ for polynomial chaos expansions is polynomial in x, real roots may not exist for a polynomial with even order. A polynomial with odd order, on the contrary, always has at least one real root for $y \in [-\infty, \infty]$. Consequently, the new constraint is added to formulation (P1) when the number of terms in the expansion is odd (for example, the highest order in the three-term polynomial chaos expansion is two). The new optimization problem is formulated as:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{M} f_i^2 && \text{(P2)} \\
\text{s.t.} \quad & m_{iT} - m_{iA} = f_i \; ; \; i = 1, \ldots, M \\
& \text{if M is odd and } a_{M-1} > 0 \\
& \sum_{i=0}^{M-1} a_i \, \Psi_i(x) = L \\
& \text{if M is odd and } a_{M-1} < 0 \\
& \sum_{i=0}^{M-1} a_i \, \Psi_i(x) = U
\end{aligned}$$

where $\Psi_i(x)$ is the i-th polynomial chaos or Hermite polynomial. It should be noted that the formulation (P2) can be used for a random variable not only with an analytical form of distribution, but also with an empirical form of distribution. The resulted approximation of the empirical distribution will give an optimal approximation to the underlying distribution of data.

As an example, for the problem of approximating a log-normal random variable, one

can formulate the following optimization scheme

$$\begin{aligned} \min \quad & \sum_{i=1}^{3} f_i^2 \\ \text{s.t.} \quad & \mu_T - \mu_A = f_1 \\ & \sigma_T^2 - \sigma_A^2 = f_2 \\ & \gamma_{1T} - \gamma_{1A} = f_3 \\ & a_2 \geq 0 \\ & a_0 + a_1\, x + a_2\, (x^2 - 1) = 0 \end{aligned}$$

to obtain a three-term polynomial chaos expansion.

A similar approach can be used to approximate a vector of N random variables which appears when those random variables are correlated to each other. The optimization formulation becomes

$$\begin{aligned} \min \quad & \sum_{j=1}^{N} \sum_{i=1}^{M_j} f_{i,j}^2 \qquad \text{(P3)} \\ \text{s.t.} \quad & \text{for } j = 1, \ldots, N: \\ & m_{i,jT} - m_{i,jA} = f_{i,j} \;;\; i = 1, \ldots, M_j \\ & \text{if } \mathrm{M_j} \text{ is odd and } \mathrm{a_{M-1,j}} > 0 \\ & \sum_{i=0}^{M_j - 1} a_{i,j}\, \Psi_i(\bar{x}) = L_j \\ & \text{if } \mathrm{M_j} \text{ is odd and } \mathrm{a_{M-1,j}} < 0 \\ & \sum_{i=0}^{M_j - 1} a_{i,j}\, \Psi_i(\bar{x}) = U_j \end{aligned}$$

where $\Psi_i(\bar{x})$ is the i-th multi-dimensional polynomial chaos.

4.4 Methods for generating probability density function

4.4.1 Analytical method

Theorem 4.1 provides a mechanism to generate the density function of a function of a random variable with known distribution. This theorem can be extended to cover functions of random variables. The following theorem [120] describes the procedure for two functions of two random variables.

Theorem 4.2 *To find the joint density of the random variables*

$$\begin{aligned} z &= g(x, y) \\ w &= h(x, y) \end{aligned}$$

at specific value of z and w in terms of the joint density of x and y, the real roots of the following system

$$g(x, y) = z$$

$$h(x,y) \;=\; w$$

are calculated, and denoted by (x_n, y_n). *The joint density of* z *and* w *becomes*

$$f_{zw}(z,w) \;=\; \frac{f_{xy}(x_1,y_1)}{|J(x_1,y_1)|} + \cdots + \frac{f_{xy}(x_n,y_n)}{|J(x_n,y_n)|} + \cdots$$

where

$$J(x,y) \;=\; \begin{vmatrix} \frac{\partial z}{\partial x} & \frac{\partial z}{\partial y} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} \end{vmatrix}$$

is the determinant of the Jacobian of transformation.

A further extension to any number of functions and random variables is similar to the above theorem.

As an example for this procedure, consider two random variables, z and w, expanded into polynomial chaos expansions up to the linear terms. All coefficients in the expansions are known.

$$\begin{aligned} z &= z_0 + z_1 x + z_2 y \\ w &= w_0 + w_1 x + w_2 y, \end{aligned}$$

where x and y denote two standard independent Gaussian distributed random variables. It means that the joint distribution of x and y can be written as follows:

$$f_{xy}(x,y) \;=\; \frac{1}{2\pi} \exp\left(-\frac{x^2+y^2}{2}\right).$$

The determinant of the Jacobian for this system becomes

$$J(x,y) \;=\; \begin{vmatrix} z_1 & z_2 \\ w_1 & w_2 \end{vmatrix}$$

from which its absolute value is

$$|J(x,y)| \;=\; |z_1 w_2 \;-\; z_2 w_1|.$$

When $w_1 z_2 - w_2 z_1 \neq 0$, the system has one and only one solution

$$\begin{aligned} x &= \frac{w_2\,(z_0 - z) - z_2\,(w_0 - w)}{w_1 z_2 - w_2 z_1} \\ y &= \frac{w_1\,(z_0 - z) + z_1\,(w_0 - w)}{w_1 z_2 - w_2 z_1}, \end{aligned}$$

therefore, the joint density function of z and w can be written as follows.

$$\begin{aligned} f_{zw}(z,w) \;=\; & \frac{1}{|z_1 w_2 \;-\; z_2 w_1|} \times \\ & f_{xy}\left(\frac{w_2\,(z_0 - z) - z_2\,(w_0 - w)}{w_1 z_2 - w_2 z_1}, \frac{w_1\,(z_0 - z) + z_1\,(w_0 - w)}{w_1 z_2 - w_2 z_1}\right) \end{aligned}$$

On the other hand, $w_1 z_2 - w_2 z_1 = 0$ indicates that z and w are ± 1 correlated, or in other words, the value of z can be calculated from the value of w and vice-versa, and the joint density function of z and w contains a delta function, $\delta(z - w)$.

Random variables with higher order polynomial chaos expansions require computer algebra software to find the real roots and to calculate the density function automatically. Therefore, a program in the **Mathematica** language (it is given in appendix C) was written for computing the joint density function of two random variables expanded into polynomial chaos expansions. It numerically calculates the value of the density function for several points and stores the results into a file.

The previous theorem only describes a procedure for a fully determined system (the number of dependent variables equals to the number of independent variables). In practical situation, one may also want to compute the density function of one or two random variables which are functions of more than one or two other random variables respectively. This underdetermined system can be transformed to a determined system by adding some dummy dependent variables. Then theorem 4.2 can again be used. To generate the joint density function of the original dependent variables, the result from applying such a theorem is integrated over all dummy dependent variables [120].

When the random variables in the function are independent such as in the polynomial chaos expansions, the calculation of the joint density becomes simpler. The following lemma provides the procedure.

Lemma 4.1 *To find the joint density of the random variables*

$$\begin{aligned} p &= g(x, y, z) \\ q &= h(x, y, z) \end{aligned}$$

at a specific value of p and q in terms of the joint density of x, y, and z, and x, y, and z are independent, the real roots of the following system

$$\begin{aligned} g(x, y, z) &= p \\ h(x, y, z) &= q \end{aligned}$$

are calculated, and denoted by $(x_n, y_n; z)$. The joint density of p and q becomes

$$f_{pq}(p, q) = \int_{-\infty}^{\infty} \left(\frac{f_x(x_1; z) f_y(y_1; z)}{|J(x_1, y_1; z)|} + \cdots + \frac{f_x(x_n; z) f_y(y_n; z)}{|J(x_n, y_n; z)|} + \cdots \right) f_z(z)\, dz$$

where

$$J(x, y; z) = \begin{vmatrix} \frac{\partial p}{\partial x} & \frac{\partial p}{\partial y} \\ \frac{\partial q}{\partial x} & \frac{\partial q}{\partial y} \end{vmatrix}$$

is the determinant of the Jacobian of transformation.

It should be noted that the integration requirement in such a procedure always slows down the symbolic computation process, especially when the degree of underdetermination is high. Therefore, the analytical method is practical only for a determined system with low-order of polynomial chaos expansion.

Two examples of the application of the subroutine **CalDF** in appendix C are considered below.

Example 4.4.1.1

Consider that a random variable z is represented by the following polynomial chaos expansion:

$$z = 25.0 + 3.0\,\xi - 1.0\,(\xi^2 - 1) + 0.5\,(\xi^3 - 3\xi).$$

Then subroutine **CalDF** generates the probability distribution of z as shown in figure (4-5).

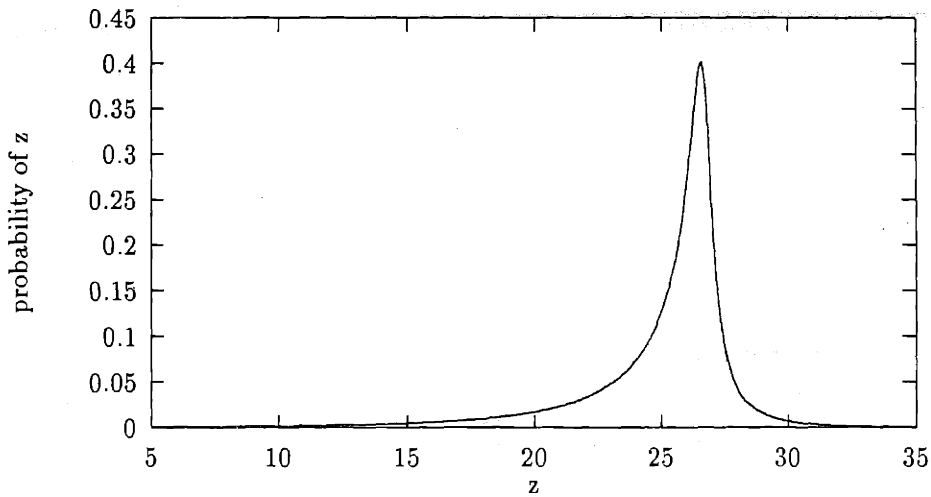


Figure 4-5: First example for the application of **CalDF**

Example 4.4.1.2

Similarly, consider that a random variable z is represented by the following polynomial chaos expansion:

$$z = 25.0 + 3.0\,\xi_1 + 2.0\,\xi_2.$$

Then subroutine **CalDF** generates the probability distribution of z as shown in figure (4-6).

There are two important points that should be made at this moment. First, theorems 4.1 and 4.2 cannot be used to calculate the density function in the region where the dependent variables have constant value because we have an infinite number of roots. In this case, we need to use the method of delta function according to the following theorem [130].

Theorem 4.3 *If $y = g(x)$, then the random variable y at any value of* x *of the random variable x has the unique possible value $g(\mathrm{x})$ and the conditional probability of this value at a given* x *is 1. Therefore, the conditional density of the random variable y at $x = \mathrm{x}$ represents the δ function:*

$$f_{y|\mathrm{x}}(y|\mathrm{x}) = \delta(y - g(\mathrm{x})).$$

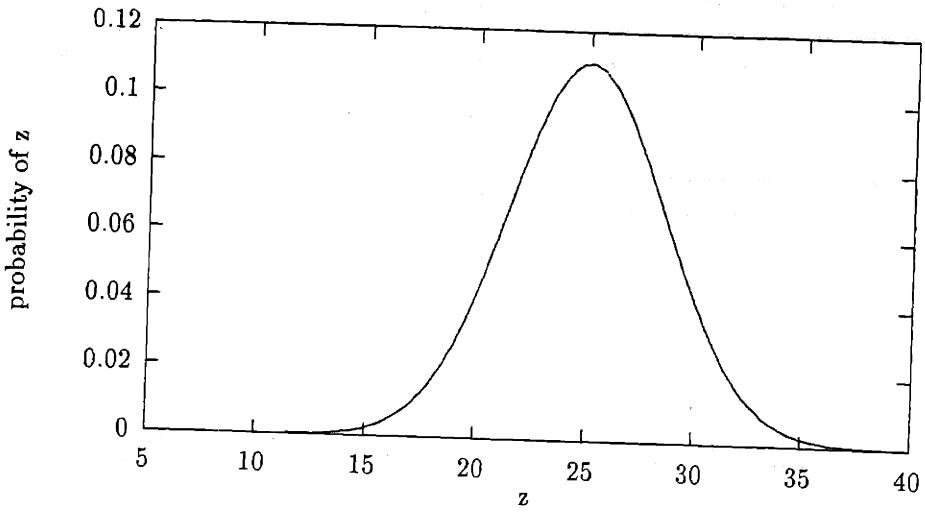


Figure 4-6: Second example for the application of **CalDF**

Consequently the joint density of the random variables x and y is determined by

$$f_{xy}(x,y) \;=\; f_x(x)\;\delta(y-g(x)).$$

Integrating with respect to x, we find the density of the random variable y:

$$f_y(y) \;=\; \int_{-\infty}^{\infty} f_x(x)\;\delta(y-g(x))\;dx.$$

Since we used the polynomial chaos expansion for representing a random variable, we almost do not have cases of constant values.

Moreover, if the dimension of the vector of dependent variables is greater than the dimension of the vector of independent variables, then the distribution of dependent random variables may or may not have a non-singular distribution [130].

If we know the characteristic function of independent random variables, then we can use the method of characteristic function to generate the distribution of dependent random variables. The following theorem provides the mechanism for conducting such a task [130].

Theorem 4.4 *Let us consider a random variable $y = g(x)$ where $g(x)$ is any measurable function. According to the definition of characteristic function, we have*

$$\Phi_y(\lambda) \;=\; \int_{-\infty}^{\infty} e^{i\lambda g(x)}\;f_x(x)\;dx,$$

where $\Phi_y(\lambda)$ is the characteristic function of y and $f_x(x)$ is the density function of the random variable x. After determining in such a way the characteristic function of y, one may find its density function by formula:

$$f_y(y) \;=\; \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\lambda y}\;\Phi_y(\lambda)\;d\lambda.$$

Examples for the application of this theorem can be found elsewhere [120, 130].

All these analytical methods are only suitable for small scale problems, involving a small number of dependent and independent random variables and analytically tractable functions. Therefore we need to use numerical approximation methods for generating the probability density function of an arbitrary random variable approximated by the polynomial chaos expansion.

4.4.2 Moments methods

Since analytical methods require the computation of all real roots of the function at each value of the dependent random variables or to generate its Fourier transfrom for every frequency and their inverse for every value of dependent random variables, the computational time can be prohibitively expensive. One way to avoid this problem is to use partial information from the random variable such as its moments. Based on that partial information, we can then construct an approximation to the probability density function of random variables.

Several moments methods have been developed for several decades [184, 122, 42, 70]. These methods can be categorized into five classes. These classes will be briefly discussed in the following text.

Von Mises step function approximation

We state the following theorem [184] which can be used as a machinery to produce the step function approximation of a distribution function of a random variable x. The input to this procedure is the set of moments of x.

Theorem 4.5 *If $M_0, M_1, \ldots, M_{2m-1}$ are the moments of a cumulative distribution $F_x(x)$, which increases at m points at least, then there exists one and only one m-step distribution $G_x(x)$ which has these moments. The abscissas a_i of the steps, $i = 1, 2, \ldots, m$ are the m real roots of equation (4.11) where the c_j are the solution of equation (4.10); the heights A_i of the m steps follow from equation (4.12). The m steps are in the interior of the smallest interval which contains all points of increase of $f_x(x)$. Either $F_x(x) \equiv G_x(x)$, or $F_x(x)$ crosses each step of $G_x(x)$.*

$$\begin{array}{lllllll} c_0M_0 & + c_1M_1 & + \cdots + & c_{m-1}M_{m-1} & = & -M_m \\ c_0M_1 & + c_1M_2 & + \cdots + & c_{m-1}M_m & = & -M_{m+1} \\ \vdots & & & \vdots & = & \vdots \\ c_0M_{m-1} & + c_1M_m & + \cdots + & c_{m-1}M_{2m-2} & = & -M_{2m-1} \end{array} \tag{4.10}$$

$$x^m + c_{m-1}x^{m-1} + c_{m-2}x^{m-2} + \cdots + c_1x + c_0 = 0 \tag{4.11}$$

$$M_k - \sum_{i=1}^{m} a_i^k A_i = 0, \quad k = 0, 1, 2, \cdots, m-1 \tag{4.12}$$

To use this method we have to solve two systems of linear equations, each has m unknowns, and to calculate all real roots of a polynomial of degree m. There are also some conditions on the value of moments that have to be fulfilled so that the distribution $F_x(x)$

increases at more than $m-1$ points [184]. A simple example of the application of Von Mises method for approximating a Gaussian distributed random variable is considered below.

Example 4.4.2.1

The first six moments of a standardized Gaussian distributed random variable are given, $M_0 = 1, M_1 = 0, M_2 = 1, M_3 = 0, M_4 = 3, M_5 = 0$. The goal is to approximate the distribution of this random variable by using the Von Mises approximation method. First, we have to check whether the distribution increases at more than $m-1$ points. Since for our example $2m-1 = 5$, then $m = 3$. If following conditions are satisfied

$$
\begin{aligned}
M_0 &> 0, \\
\begin{vmatrix} M_0 & M_1 \\ M_1 & M_2 \end{vmatrix} &> 0, \\
\begin{vmatrix} M_0 & M_1 & M_2 \\ M_1 & M_2 & M_3 \\ M_2 & M_3 & M_4 \end{vmatrix} &> 0,
\end{aligned}
$$

then the distribution increases at more than 2 points.

Clearly, those conditions are satisfied for our case. The next step is to solve equation (4.10), and the result is $c_0 = 0, c_1 = -3, c_2 = 0$. Substituting this result to the polynomial (4.11) we obtain the abscissas for our approximation as the roots of such a polynomial $a_1 = -\sqrt{3}, a_2 = 0, a_3 = \sqrt{3}$. The heights of the step function which calculated from equation (4.12) are $A_1 = \frac{1}{6}, A_2 = \frac{2}{3}, A_3 = \frac{1}{6}$. These results are shown in figure 4-7 along with the true Gaussian distribution.

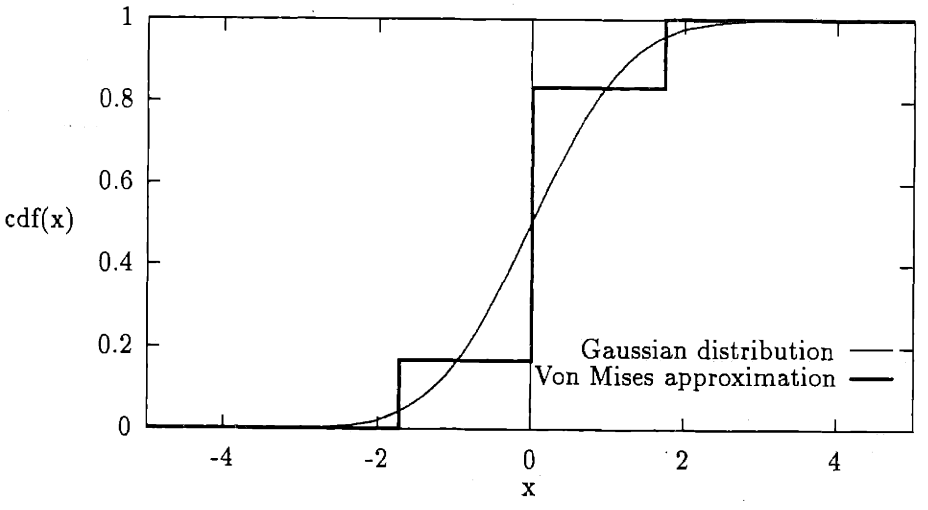


Figure 4-7: example for Von Mises approximation method

Another example of the Von Mises method can be found in Thompson and Palicio's paper [168]. They use this method for approximating the posterior distribution of system availability.

It should be pointed out that since the number of steps is half of the number of moments available, this method requires a lot of moments for approximating a smooth distribution function. In practice, it is difficult to calculate high moments accurately, therefore, what we need are methods which can sufficiently approximate a smooth distribution function using as small number of moments as possible.

Frequency curves systems

There are two common frequency curves systems: the Pearson curves and the Johnson curves systems. The Pearson curves system is developed from considering ordinary behaviors of a probability density function. On the other hand, the Johnson curves system is obtained by making specific transformations to the Gaussian probability density function. To use these methods, one has to estimate four parameters from the density function. We will discuss first the Pearson curves system and afterward move to the Johnson curves system. The description of other families of frequency distributions can be found in Ord's book [118].

For the derivation of Pearson curves system, we follow the idea from Elderton and Johnson [44]. In general, one can write the first derivative of a probability density function of a random variable as follow:

$$\frac{df_x(x)}{dx} = \frac{f_x(x)\,(x+a)}{H(x)}, \tag{4.13}$$

where $H(x)$ is an arbitrary function of x. This specific structure of the derivative comes from the knowledge that for certain values of x, namely, at the maximum, $x = -a$, and at the end of the probability density function or $f_x(x) = 0$ there is contact with the axis of x, and the value of derivative equals to zero, $\frac{df_x(x)}{dx} = 0$. Furthermore, since $H(x)$ is an arbitrary function of x one can force that $\frac{df_x(x)}{dx} \neq 0$ when neither $f_x(x) = 0$ nor $x = -a$. If $H(x)$ is expanded by Maclaurin's theorem in ascending powers of x, we have

$$\frac{df_x(x)}{dx} = \frac{f_x(x)\,(x+a)}{b_0 + b_1 x + b_2 x^2 + \ldots}. \tag{4.14}$$

The truncated version of equation (4.14) up to the b_2 term is the starting point for generating Pearson curves system. The Pearson curves system has therefore four coefficients, a, b_0, b_1, b_2, that have to be calculated from the knowledge of the first four moments. The relationship between these coefficients and moments values can be derived by considering that

$$x^r f_x(x) \to 0 \quad \text{as} \quad x \to \pm\infty \quad \text{for} \quad 0 \leq r \leq 3. \tag{4.15}$$

First, we multiply each side of equation (4.14) by x^r, and integrate it with respect to x, we have

$$\int_{-\infty}^{\infty} x^r\,(b_0 + b_1 x + b_2 x^2)\,\frac{df_x(x)}{dx}\,dx = \int_{-\infty}^{\infty} f_x(x)\,(x+a)\,x^r\,dx.$$

Integrating the left-hand side by parts and treating $\frac{df_x(x)}{dx}$ as one part, and the right-hand side as the sum of two functions, we obtain

$$\begin{aligned} x^r\ (b_0 + b_1 x + b_2 x^2)\ f_x(x)\ \Big|_{-\infty}^{\infty} \quad &- \quad \int_{-\infty}^{\infty} \Big(r b_0 x^{r-1} + \\ (r+1) b_1 x^r + (r+2) b_2 x^{r+1} \Big)\ f_x(x)\ dx \quad &= \quad \int_{-\infty}^{\infty} f_x(x) x^{r+1}\ dx + \int_{-\infty}^{\infty} f_x(x) a x^r\ dx. \end{aligned}$$

From condition (4.15), we have the following relationship

$$-r b_0 m_{r-1} - ((r+1) b_1 + a)\, m_r - ((r+2) b_2 + 1)\, m_{r+1} \quad = \quad 0; \quad r = 0, 1, 2, 3. \quad (4.16)$$

where $m_r = \int_{-\infty}^{\infty} x^r f_x(x)\ dx$, and $m_{-1} = 0$. Without loss of generality we can assume that $m_1 = 0$, and the coefficients of the Pearson curves system are

$$\begin{aligned} a &= -b_1 \\ b_0 &= -\frac{4\beta_2 - 3\beta_1}{10\beta_2 - 12\beta_1 - 18}\ m_2 \\ b_1 &= -\frac{\beta_2 + 3}{10\beta_2 - 12\beta_1 - 18}\ \sqrt{\beta_1 m_2} \\ b_2 &= -\frac{2\beta_2 - 3\beta_1 - 6}{10\beta_2 - 12\beta_1 - 18} \end{aligned}$$

where $\beta_1 = \frac{m_3^2}{m_2^3}$ and $\beta_2 = \frac{m_4}{m_2^2}$.

After one obtains the value of coefficients in the Pearson curves system, the probability density function can be obtained by integrating the following equation

$$\frac{d \log(f_x(x))}{dx} \quad = \quad \frac{x + a}{b_0 + b_1 x + b_2 x^2}.$$

The form of the integral depends on the particular values of the coefficients of x in the denominator, and obviously, the criterion for fixing the form in a particular case is the same as that for the nature of the roots of the equation $b_0 + b_1 x + b_2 x^2$, that is $\frac{b_1^2}{4 b_0 b_2}$, or from the relationship with moments values we have

$$\kappa \quad = \quad \frac{\beta_1\ (\beta_2 + 3)^2}{4\ (2\beta_2 - 3\beta_1 - 6)\ (4\beta_2 - 3\beta_1)},$$

where κ denotes the criterion for having a particular form of probability density function.

There are three main types (I, IV, VI), and ten transition types (N, II, III, V, VII, VIII, IX, X, XI, XII) in the Pearson curves system. The main types represent an area on the (β_1, β_2) plane, on the other hand, the transition types represent either a line or a point on such a plane [72]. The following list of criteria follows the same notation as the Pearson classification.

Type N: $\kappa = 0,\ \beta_1 = 0,\ \beta_2 = 3$

Type I: $\kappa < 0$

Type II: $\kappa = 0,\ \beta_1 = 0,\ \beta_2 < 3$

Type III: $2\beta_2 - 3\beta_1 - 6 = 0$

Type IV: $0 < \kappa < 1$

Type V: $\kappa = 1$

Type VI: $\kappa > 1$

Type VII: $\kappa = 0,\ \beta_1 = 0,\ \beta_2 > 3$

Type VIII: $\kappa < 0,\ \lambda = 0,\ 5\beta_2 - 6\beta_1 - 9 < 0$

Type IX: $\kappa < 0,\ \lambda = 0,\ 5\beta_2 - 6\beta_1 - 9 > 0,\ 2\beta_2 - 3\beta_1 - 6 < 0$

Type X: $\beta_1 = 4,\ \beta_2 = 9$

Type XI: $\kappa > 1,\ \lambda = 0,\ 2\beta_2 - 3\beta_1 - 6 > 0$

Type XII: $5\beta_2 - 6\beta_1 - 9 = 0$

where $\lambda = \frac{(4\beta_2-3\beta_1)(10\beta_2-12\beta_1-18)^2-\beta_1(\beta_2+3)^2(8\beta_2-9\beta_1-12)}{(3\beta_1-2\beta_2+6)\beta_1(\beta_2+3)^2+4(4\beta_2-3\beta_1)(3\beta_1-2\beta_2+6)}$. A table which shows the form of the probability density function for each type can be seen in Elderton and Johnson's book [44]. An extension of the Pearson curves system for more than one dimension can be found elsewhere [73, 162]. A simple example of Pearson curves system for approximating a Gaussian probability density function is considered below.

Example 4.4.2.2

The first four moments of a standardized Gaussian random variable are $0, 1, 0, 3$. The values of β_1, β_2, and κ are therefore $0, 3$, and 0 respectively. We can calculate the coefficients in the Pearson system, and we have $a = -b_1 = 0, b_0 = -1, b_2 = 0$. Hence, our differential equation becomes

$$\frac{d\log(f_x(x))}{dx} = -x,$$

and as a result

$$f_x(x) = f_0 \exp(-\frac{x^2}{2})$$

where f_0 is a constant such that $\int_{-\infty}^{\infty} f_x(x)\,dx = 1$. In this case $f_0 = \frac{1}{\sqrt{2\pi}}$.

From the above list of criteria, we see that this form of density function belongs to type N, which stands for the Normal distribution function.
In general, we can therefore use that list and the related table [44] to generate probability density functions under consideration.

The Johnson curves system uses the translation method to generate an approximation of a probability density function. The concept of translation method was developed by Edgeworth for transforming a non-Gaussian random variable to a Gaussian one. Figure 4-8 shows the result in terms of the probability density function from an arbitrary transformation process. In general, we can write the transformation from one random variable to another as follows:

$$z = \gamma + \delta g\left(\frac{x-\xi}{\lambda}\right).$$

Without loss of generality, we can simplify the above transformation by adding a new transformation:

$$x = \xi + \lambda y,$$

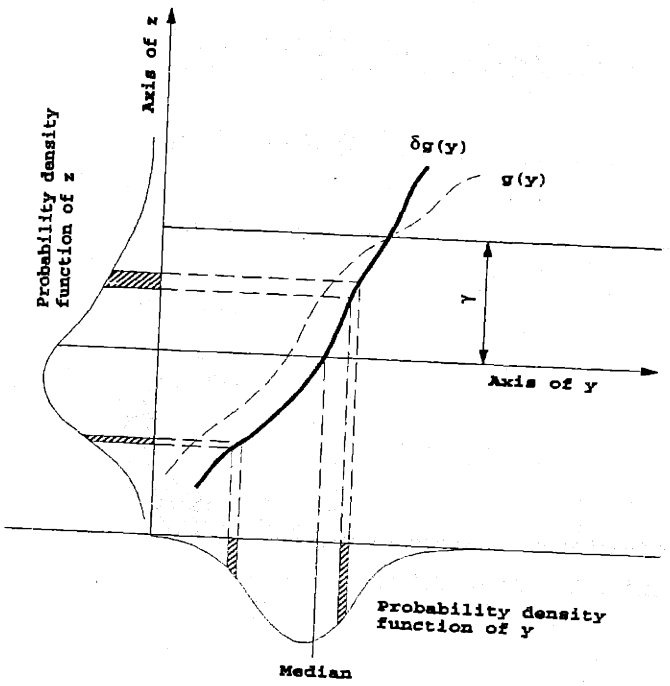


Figure 4-8: General transformation of a random variable, $z = \gamma + \delta g(y)$

we have

$$z = \gamma + \delta\, g(y). \tag{4.17}$$

We see that there are four parameters which should be calculated from the knowledge of the first four moments of x. It should be pointed out that we can use other methods for fitting those parameters, such as the method of percentile points or the method of maximum likelihood [71]. In this subsection, we will only use the moments values.

The problem now is to choose the cumulative distribution function of z such that it is simple to calculate. The existence of many tabulated values for the cumulative distribution function of a Gaussian random variable suggests us to choose z as a Gaussian random variable. Another problem is to choose the functional form of $g(y)$. For convenience, this function should possess the following properties [71]:

1. It should be a monotonic function of y.
2. It should be simple in form and easy to calculate.
3. The range of values of $g(y)$ in the domain of y should correspond to the actual range of possible values of z, in this case $-\infty$ to $+\infty$.

The reasons for item two and three are clear. The first item is necessary for having a simple relationship of probability density functions between z and y,

$$f_y(y) = \left|\delta\, \frac{dg(y)}{dy}\right| f_z(\gamma + \delta g(y)), \tag{4.18}$$

and similarly for $f_x(x)$,

$$f_x(x) = \frac{1}{|\lambda|} f_y\left(\frac{x-\xi}{\lambda}\right).$$

Johnson used three different forms of $g(y)$ depending on the range of the random variable y [71]:

$$\begin{aligned} z &= \gamma + \delta \ln(y), \quad y > 0; \quad S_L \text{ type} \\ z &= \gamma + \delta \ln\left(\frac{y}{1-y}\right), \quad 0 < y < 1; \quad S_B \text{ type} \\ z &= \gamma + \delta \sinh^{-1}(y); \quad S_U \text{type}. \end{aligned}$$

We can therefore choose the right form of transformation if we know the boundedness of the domain of the random variable y. When we do not have such a knowledge we can use the following criteria to choose a suitable form of transformation.

- S_L type lies on a curve on the (β_1, β_2) plane defined by the parametric equations:

$$\begin{aligned} \beta_1 &= (\omega - 1)(\omega + 2)^2; \quad \sqrt{\beta_1} > 0 \\ \beta_2 &= \omega^4 + 2\omega^3 + 3\omega^2 - 3 \end{aligned}$$

  where $\omega = \exp(\delta^{-2})$ and β_1, β_2 have similar definitions as in the Pearson curves system discussion.

- S_B type includes points on (β_1, β_2) plane which its lower boundary is S_L line and its upper boundary defined as:

$$\beta_2 - \beta_1 - 1 = 0.$$

- S_U type includes points on (β_1, β_2) plane which its upper boundary is S_L line.

By analogy to the Pearson curves system, the S_L type is the transition type and others are the main types of the Johnson curves system. To fit the parameters δ, γ, we need the relationships between moments values and those parameters. Since the S_L type is confined in a curve on (β_1, β_2) plane, the number of parameters and corresponding moments needed is in fact only three (in this case, $\lambda = 1$). Therefore we need only to solve the following equations.

$$\begin{aligned} m_1(y) &= m_1(x) \\ m_2(y) &= m_2(x) - 2\xi m_1(x) \\ m_3(y) &= m_3(x) - 3\xi m_2(x) + 3\xi^2 m_1(x) - \xi^3, \end{aligned}$$

where $m_n(y) = \exp\left(\frac{n^2 - n\gamma}{2\delta^2}\right)$. If we know the value of the first three moments of x, $m_i(x)$, $i = 1, \ldots, 3$, we can compute δ, γ, ξ, and the probability density function of x becomes

$$f_x(x) = \frac{\delta}{\sqrt{2\pi}(x-\xi)} \exp\left(-\frac{1}{2}(\gamma + \delta \ln(x-\xi))^2\right); \quad x > \xi$$

For S_U type system, the knowledge of four moments of y is used to compute the parameters. Johnson gave a chart to infer the value of δ, γ from the knowledge of β_1, β_2, or if one wants to compute those parameters numerically the following equations can be used [71].

$$\begin{aligned}
\beta_1 &= \frac{\mu_3^2}{\mu_2^3} \\
\beta_2 &= \frac{\mu_4}{\mu_2^2} \\
\mu_2 &= \frac{1}{2}(\omega-1)(\omega\cosh(2\Omega)+1) \\
\mu_3 &= -\frac{1}{4}\omega^{\frac{1}{2}}(\omega-1)^2(\omega(\omega+2)\sinh(3\Omega)+3\sinh(\Omega)) \\
\mu_4 &= \frac{1}{8}(\omega-1)^2\Big(\omega^2(\omega^4+2\omega^3+3\omega^2-3)\cosh(4\Omega)+ \\
&\quad 4\omega^2(\omega+2)\cosh(2\Omega)+3(2\omega+1)\Big),
\end{aligned}$$

where $\omega = \exp(\delta^{-2})$, and $\Omega = \frac{\gamma}{\delta}$. After obtaining the values of ω, Ω or δ, γ, one can compute the values of ξ, λ from the knowledge of the mean value, $mu_1(x)$, and variance, $\mu_2(x)$, of x as follows:

$$\begin{aligned}
\lambda &= \frac{\mu_1(x)}{-\omega^{\frac{1}{2}}\sinh(\Omega)} \\
\lambda^2\mu_2 &= \xi^2 - 2\xi\mu_1(x) + \mu_2(x).
\end{aligned}$$

As a result, the probability density function for this type can be written as

$$f_x(x) = \frac{1}{\lambda}\frac{\delta}{\sqrt{2\pi}}\frac{1}{\sqrt{\left(\frac{x-\xi}{\lambda}\right)^2+1}}\exp\left(-\frac{1}{2}\left(\gamma+\delta\ln\left(\frac{x-\xi}{\lambda}+\sqrt{\left(\frac{x-\xi}{\lambda}\right)^2+1}\right)\right)^2\right).$$

Since the expression of moments for S_B type system is complicated, the application of moments method to fit the parameters in the transformation is not recommended. The moments values involve infinite series in hyperbolic functions of parameters. In the case that both end-points of the distribution of x are known then one can use the maximum likelihood method to fit the parameters, otherwise the method of percentiles is the most convenient to use [71]. In the case that we are able to compute the parameters, the probability density function of x is given as

$$f_x(x) = \frac{1}{\lambda}\frac{\delta}{\sqrt{2\pi}}\frac{\lambda^2}{(x-\xi)(\xi+\lambda-x)}\exp\left(-\frac{1}{2}\left(\gamma+\delta\ln\left(\frac{x-\xi}{\xi+\lambda-x}\right)\right)^2\right); \quad \xi < x < \xi+\lambda.$$

The Johnson curves system, S_L, S_U, S_B, together with the Gaussian curve combine to give a variety of shapes of curve as that provided by the Pearson curves system in general use. Johnson [71] also pointed out that except for discrepancies at the ends of distribution, curves of S_L, S_U, and S_B agree generally with Pearson curves having the same, or nearly the same the first four moments, and the use of Johnson curves system may sometimes be considered simply as a convenient aid in computing rough numbers for subrange distribution of the Pearson curves system. Similar to the Pearson curves system, Johnson curves system

can be extended to approximate a bivariate probability density function [73].

Orthogonal expansions

There are several important characteristics of a probability density function which should be considered before one uses orthogonal functions to fit such a density function. They are the range and the modality of the probability density function. Another crucial problem in this process is to determine the existence of a distribution function given a finite number of its moments. Therefore, we will discuss several theorems [66] related to the existence of a distribution, the maximum number of moments required, and the unimodality requirement.

The following theorems provides the criteria or conditions for the existence of a distribution function given a number of its moments. First, we deal with the distribution function defined over the entire real line $(-\infty, \infty)$.

Theorem 4.6 *A necessary and sufficient condition that there should exist at least one non-decreasing function $F_x(x)$ such that*

$$m_j = \int_{-\infty}^{\infty} x^j \, dF_x(x), \quad j = 0, 1, 2, \ldots, 2n-1$$

is that the quadratic form

$$\sum_{i,j=0}^{n} m_{i+j} a_i a_j$$

be positive (definite or semidefinite), where a_i is any real number. In other words, the sequence $m_0, m_1, \ldots, m_{2n-1}$ is positive.

Next consider the case where the distribution function of interest is defined over the positive real half line $(0, \infty)$.

Theorem 4.7 *A necessary and sufficient condition that there should exist a non-decreasing function $F_x(x)$ such that*

$$m_j = \int_{0}^{\infty} x^j \, dF_x(x), \quad j = 0, 1, 2, \ldots, n$$

is that the quadratic forms

$$\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} m_{i+j} a_i a_j$$

and

$$\sum_{i=0}^{\lfloor \frac{n-1}{2} \rfloor} \sum_{j=0}^{\lfloor \frac{n-1}{2} \rfloor} m_{i+j+1} a_i a_j$$

should be positive (definite or semidefinite).

A theorem for determining the positivity of a matrix of moment values is required to make both above theorems practical. The following theorem describes a set of criteria for checking the positivity condition.

Theorem 4.8 *A set of necessary and sufficient conditions for the form*

$$\sum_{i=0}^{n}\sum_{j=0}^{n} m_{i+j}a_i a_j \;=\; a^T M a$$

to be positive is that all the principal minors of M should be positive,

$$\begin{aligned}
v_0 &\geq 0,\\
\begin{vmatrix} v_0 & v_1 \\ v_1 & v_2 \end{vmatrix} &\geq 0,\\
&\vdots\\
\begin{vmatrix} v_0 & v_1 & \cdots & v_n \\ v_1 & v_2 & \cdots & v_{n+1} \\ \vdots & & \cdots & \vdots \\ v_n & v_{n+1} & \cdots & v_{2n} \end{vmatrix} &\geq 0,
\end{aligned}$$

From these theorems, one can determine if there is a maximum number of moments that specifies the unknown probability density function. It is possible because the theorems tell us that n moments specify a distribution if and only if the corresponding quadratic forms are positive which is true if and only if the resulting determinants are non-negative [66]. Therefore, given a number of moments, N, one can test the above criteria successively until for some value of $n \leq N$ one of the determinants is negative. If that happens, the maximum number of moments which constitutes a description of the unknown probability density function is $n-1$, otherwise one may use all N moments.

From the knowledge of the moment values, one could also infer the modality of the unknown probability density function. This modality property is an important factor because orthogonal polynomial expansion methods can easily produce a multimodal distribution function which sometimes is not desirable based on the physical consideration of the data or the practicality of the approximation. The following theorems [66] discuss the conditions for having a unimodal distribution function. First we will see the case of a distribution function with the entire real line support.

Theorem 4.9 *A necessary and sufficient condition that the distribution function $F_x(s)$ with mode L and*

$$m_j \;=\; \int_{-\infty}^{\infty} x^j dF_x(x), \quad j = 0, 1, \ldots, n$$

be unimodal is that there exist a real number L such that

$$m_1 - \sqrt{3\left(m_2 - m_1^2\right)} \;\leq L \leq\; m_1 + \sqrt{3\left(m_2 - m_1^2\right)}$$

and that the roots r_1, r_2, and r_3 of the cubic y,

$$y \;=\; ar^3 + br^2 + cr + d \;\geq\; 0,$$

where

$$a \;=\; 4m_3 - 12m_1m_2 + 8m_1^3$$

$$
\begin{aligned}
b &= 8m_1m_3 + 9m_2^2 - 12m_1^2m_2 - 5m_4 \\
c &= 10m_1m_4 - 12m_2m3 - 16m_1^2m_3 + 18m_1m2^2 \\
d &= 15m_2m_4 + 48m_1m_2m_3 - 27m_2^3 - 16m_3^2 - 20m_1^2m_4
\end{aligned}
$$

satisfy the conditons:

$$r_1 \leq L$$

if y has only one real root and $a > 0$, or

$$r_1 \geq L$$

if y has only one real root and $a < 0$; or if all roots are real ($r_1 \leq r_2 \leq r_3$)

$$r_3 \leq L \quad or \quad r_1 \leq L \leq r_2$$

when $a > 0$, or

$$r_3 \geq L \quad or \quad r_2 \leq L \leq r_3$$

when $a < 0$.
In the case where $a = 0$, the real roots $r_1 \leq r_2$ of the remaining quadratic must satisfy the following conditions:

$$r_1 \leq L \leq r_2$$

when $b < 0$, or

$$r_1 \geq L \quad or \quad r_2 \leq L$$

when $b > 0$, or if both roots are complex

$$b > 0$$

must hold.
In the case that both a and b are zero, L must satisfy

$$
\begin{aligned}
L &\geq \frac{-d}{c}; \quad c > 0, or \\
L &\leq \frac{-d}{c}; \quad c < 0.
\end{aligned}
$$

In the case when a, b, and c are zero,

$$d \geq 0$$

must hold.

Similarly, the following theorem descibes the condition for a distribution function with the half real line support.

Theorem 4.10 *A necessary and sufficient condition that the distribution function $F_x(x)$*

with mode L and

$$m_j = \int_0^\infty x^j dF_x(x), \quad j = 0, 1, \ldots, n$$

be unimodal is that there exist a real number L which satisfies theorem 4.9 and also satisfies all of the following relations:

1. $L \leq 2m_1$
2. *The roots $p_1 \leq p_2$ of the quadratic z*

$$z = sp^2 + tp + u \geq 0,$$

where

$$\begin{aligned} s &= 3m_2 - 4m_1^2 \\ t &= 6m_1m_2 - 4m_3 \\ u &= 8m_1m_3 - 9m_2^2 \end{aligned}$$

satisfy one of the conditions:

(a) Both roots are real

$$\begin{aligned} p_1 \leq L \leq p_2; \quad s < 0, \text{or} \\ p_1 \geq L \quad \text{or} \quad p_2 \leq L; \quad s > 0. \end{aligned}$$

(b) Both roots ar complex

$$2\left(3m_2 - 4m_1^2\right) > 0.$$

(c) In the case that $s = 0$

$$\begin{aligned} L \geq \frac{-u}{t}; \quad t > 0, \text{or} \\ L \leq \frac{-u}{t}; \quad t < 0. \end{aligned}$$

(d) In the case that $s = t = 0$

$$u \geq 0$$

3. *The roots r_1, r_2 and r_3 of the cubic y,*

$$y = ar^3 + br^2 + cr + d \geq 0,$$

where

$$\begin{aligned} a &= 27m_2^3 + 16m_3^2 + 20m_1^2m_4 - 15m_2m_4 - 48m_1m_2m_3 \\ b &= 18m_2m_5 + 30m_1m_2m_4 + 30m_1m_3^2 - 36m_2^2m_3 - 20m_3m_4 - 24m_1^2m_5 \\ c &= 48m_2m_3^2 + 36m_1m_2m_5 + 25m_4^2 - 24m_3m_5 - 45m_2^2m_4 - 40m_1m_3m_4 \\ d &= 48m_1m_3m_5 + 120m_2m_3m_4 - 64m_3^3 - 50m_1m_4^2 - 54m_2^2m_5, \end{aligned}$$

satisfy the corresponding relationships for r_1, r_2 and r_3 of theorem 4.9.

After we have computed the maximum number of moments required and have checked the modality of the unknown probability density function, the next task is to use a series of orthogonal polynomials to approximate the unknown probability density function based on the values of moments. A general form of orthogonal expansions of an arbitrary probability density function is given as follows.

$$f_x(x) = \alpha_x(x) \sum_{i=0}^{\infty} a_i P_i(x), \tag{4.19}$$

where $\alpha_x(x)$ is the key probability density function, $P_i(x)$ is an orthogonal polynomial derived from $\alpha_x(x)$, and a_i is an appropriate coefficient. Therefore, we first choose a parametric form of function, $\alpha_x(x)$, to estimate the probability density function $f_x(x)$, and then we can use some facility, $P_i(x)$, to adjust the parametric form in case the fit to a set of given moment values is poor.

If we know the trends or general shape of $f_x(x)$, we can use problem-specific polynomials in order to reduce the error of the truncated approximation, otherwise we can use standard orthogonal polynomials, such as Hermite, Laguerre, and others. To generate a problem-specific polynomial, one needs to choose the key probability density function $\alpha_x(x)$ that has almost similar trends to or shape of the $f_x(x)$. Therefore, $\alpha_x(x)$ should satisfy the following constraints [132]. The first moment of $\alpha_x(x)$ deviates from the first moment of $f_x(x)$ by no more than a small fraction of the standard deviation of $\alpha_x(x)$. Furthermore, $\alpha_x(x)$ must have a few of the leading moments not substantially different from the corresponding moments of $f_x(x)$. After we pick the form of $\alpha_x(x)$, we can use the following formula to produce the approximation [172].

$$P_i(x) = \frac{1}{\Delta_{i-1}} \begin{vmatrix} 1 & \alpha_1 & \cdots & \alpha_i \\ \alpha_1 & \alpha_2 & \cdots & \alpha_{i+1} \\ \vdots & & \cdots & \vdots \\ \alpha_{i-1} & \alpha_i & \cdots & \alpha_{2i-1} \\ 1 & x & \cdots & x^i \end{vmatrix}$$

$$a_i = \frac{\delta_i}{\Delta_i}$$

$$\delta_i = \begin{vmatrix} 1 & \alpha_1 & \cdots & \alpha_i \\ \alpha_1 & \alpha_2 & \cdots & \alpha_{i+1} \\ \vdots & & \cdots & \vdots \\ \alpha_{i-1} & \alpha_i & \cdots & \alpha_{2i-1} \\ 1 & m_1 & \cdots & m_i \end{vmatrix}$$

$$\Delta_i = \begin{vmatrix} 1 & \alpha_1 & \cdots & \alpha_i \\ \alpha_1 & \alpha_2 & \cdots & \alpha_{i+1} \\ \vdots & & \cdots & \vdots \\ \alpha_{i-1} & \alpha_i & \cdots & \alpha_{2i-1} \\ \alpha_i & \alpha_{i+1} & \cdots & \alpha_{2i} \end{vmatrix}$$

where α_i and m_i denote the i-th moment of $\alpha_x(x)$ and $f_x(x)$ respectively. We can eliminate some intermediate calculations of the coefficients of the polynomials and the coefficients in

the expansion by rewriting the truncated version of equation (4.19) as follows [172]:

$$f_x(x) \approx \alpha_x(x)\, R_n(x),$$

where $R_n(x) = \sum_{i=0}^{n} a_i\, P_i(x)$, and can be calculated from the equation

$$R_n(x) = -\frac{1}{\Delta_n} \begin{vmatrix} 0 & 1 & x & \cdots & x^n \\ 1 & 1 & \alpha_1 & \cdots & \alpha_n \\ m_1 & \alpha_1 & \alpha_2 & \cdots & \alpha_{n+1} \\ \vdots & & & \cdots & \vdots \\ m_n & \alpha_n & \alpha_{n+1} & \cdots & \alpha_{2n} \end{vmatrix}$$

Therefore, when we have n moments of $f_x(x)$, we can only use polynomials up to the order of n. Since the value of higher moments tends to have a relatively high error, for example increasing the number of moments beyond a certain value can detrimentally affect the quality of approximation due to word-length limitations of the computer [106], Buckland proposed the use of maximum likelihood for computing the coefficients in the expansion [22]. His algorithm can be applied to both ungrouped and grouped data, and can select the optimal sequence of polynomial terms to fit the probability density function. It should be noted however that the simplicity of moments method to fit the parameters, and the application of problem-specific polynomials for reducing the number of moments required while retaining a certain level of accuracy, may favor the use of moments method for some appropriate problems.

There are several general orthogonal expansions based on the choice of the form of $\alpha_x(x)$. Edgeworth, Gram-Charlier type A (sometimes also called Bruns-Charlier), or Longuet-Higgins series expansions use the Gaussian distribution for their key probability density function. The difference between those expansions lies on the ordering of polynomial terms [72, 117, 171], in this case Hermite polynomials. These orthogonal expansions are used for approximating the probability density function with entire real line support $(-\infty, \infty)$. For approximating a probability density function with half real line support $(0, \infty)$, one can use gamma distribution for $\alpha_x(x)$, and Laguerre polynomials for $P_i(x)$ [169]. Similarly, for representing a probability density function with bounded support, such as $(0, 1)$, one can use beta or uniform distributions for $\alpha_x(x)$, and Jacobi or Legendre polynomials for $P_i(x)$ respectively [35].

All these orthogonal expansion methods can be extended for problems with more than one dimension. Similar to one dimensional cases, the correlation functions of the joint probability density function can be implicitly defined in the joint key probability density function. Incorporating the correlation functions in the joint key probability density function will increase the convergence rate of the expansion, however, it will make the generation of polynomials more complicated [166]. As an example of approximating a multivariate probability density function, Chambers [28], and Pugachev [130] showed the application of multidimensional Edgeworth series expansion to such a problem.

Linear combination of distributions

Since equation (4.19) can be seen as a functional Taylor expansion of $f_x(x)$ around $\alpha_x(x)$, the use of orthogonal functions to approximate the deviation of $f_x(x)$ from $\alpha_x(x)$ works only when we have a small deviation. As mentioned before, one way to make the deviation small

is to choose $\alpha_x(x)$ to have similar trends and shape as $f_x(x)$. In order to more accurately mimic the trends and shape of $f_x(x)$, one can use a set of probability density functions for the key probability density function.

$$f_x(x) \approx \sum_{l=1}^{L} s_l \alpha_l(x) \sum_{i=0}^{n} a_{li} P_{li}(x)$$

where

$$\sum_{l=1}^{L} s_l = 1; \quad s_l \geq 0,$$

and $\alpha_l(x)$ is a probability density function. The optimal values of $\{s_l\}$ are determined from the knowlege of N moments of $f_x(x)$, and by minimizing the objective function:

$$\epsilon_N = \sum_{j=n+1}^{N} (\gamma_j - \hat{\gamma}_j)^2,$$

where $\gamma_j = \frac{\mu_j}{\mu_2^{j/2}}$ is the j-th dimensionless central moment of $f_x(x)$, and $\hat{\gamma}_j$ denotes its approximation which defined by equation

$$\hat{\gamma}_j = \sum_{l=1}^{L} s_l \, \hat{\gamma}_{lj}$$

where $\hat{\gamma}_{lj}$ is the j-th dimensionless central moment of $\alpha_l(x)$. For this type of optimization, we can use Lagrange equations to find the optimal point,

$$2 \sum_{k=1}^{L} \sum_{j=n+1}^{N} \hat{\gamma}_{lj} \hat{\gamma}_{kj} \, s_k + \lambda - 2 \sum_{j=n+1}^{N} \gamma_j \, \hat{\gamma}_{lj} = 0; \quad l = 1, \ldots, L$$

$$\sum_{l=1}^{L} s_l = 1,$$

where λ denotes a Lagrange multiplier of the optimization. From this set of equations, one can calculate the optimal value of $\{s_l\}$ and λ. It is clear from the Lagrange equations that if a key probability density function equals to $f_x(x)$ then $s_l = 1$ for that key probability density function, and other s_l consequently equals to zero. If each $\alpha_l(x)$ is equal to $f_x(x)$ in the second moment sense which means that each $\alpha_l(x)$ has the same mean value and variance as $f_x(x)$, and the knowledge other higher moments is used for finding the optimal values of $\{s_l\}$, we can have a similar approximation equation for the cumulative distribution function, $F_x(x)$ as for the probability density function [58]:

$$F_x(x) \approx \sum_{l=1}^{L} s_l \, F_l(x),$$

where $F_l(x)$ is the cumulative distribution function for the corresponding $\alpha_l(x)$.

One obvious problem should be solved before one uses this method is how one can choose a good set of key probability density functions. A simple procedure is first to

evaluate the dimensionless central moments of a set of common distributions, and then choose a subset of it that has about the same order of magnitude for dimensionless central moments as those of $f_x(x)$. Grigoriu [57] provided tables of dimensionless central moments for seven common distributions: Gaussian, gamma, exponential, χ^2, extreme type I and II, and Weibull distributions. It should be pointed out that it is not necessary to choose probability density functions with the same domain as of $f_x(x)$.

This method clearly can be extended to cover problems with more than one dimension. In such problems the key probability density functions will be joint probability density functions, and the moments required for finding the optimal $\{s_l\}$ will include joint moments. This idea will further be discussed in the chapter 7.

Maximum entropy method

As described in chapter 3, the maximum entropy formulation can give us the least-biased probability density function for a given set of moments or averages. Therefore, without knowing the trends or general shape of the unknown distribution function, one may use this approach to generate the least-biased approximation of it. The maximum entropy method is not limited to the information of moments of random variables, it can be used when one also has knowledge about the averages of functions of those random variables. For example, a random variable, let say x, with Cauchy distribution does not possess the moments, however, it is a maximum entropy distribution over all distributions satisfying $E\left(\ln\left(\frac{1}{1+x^2}\right)\right) = \beta$, where β is a given constant [155]. This result can be obtained from the following theorem [130].

Theorem 4.11 *If the entropy of the distribution with a density $f_0(x)$ belonging to some class of distributions C can be represented in the form*

$$H_x = -\int_{-\infty}^{\infty} f(x)\ln(f_0(x))\ dx$$

$f(x)$ being any density of the class C, then $f_0(x)$ is a unique distribution possessing the maximal entropy among all the distributions of the class C.

As an example, consider an exponential distributed random variable, this random variable has the maximum entropy distribution for random variables with domain $(0,\infty)$ and have mean value μ. The explanation can be given as follows:

$$\begin{aligned}
H_x &= -\int_0^{\infty} f_E(x)\ \ln(f_E(x))\ dx \\
&= -\int_0^{\infty} f_E(x)\ \ln(\exp(-x))\ dx \\
&= \int_0^{\infty} f_E(x)\ x\ dx \ \doteq\ \mu\ = \int_0^{\infty} f(x)\ x\ dx \\
&= -\int_0^{\infty} f(x)\ \ln(\exp(-x))\ dx \\
&= -\int_0^{\infty} f(x)\ \ln(f_E(x))\ dx
\end{aligned}$$

where $f_E(x)$ is the exponential distribution, and $f(x)$ is any distribution that has mean value μ and domain $(0,\infty)$. In other words, a probability density function $f_0(x)$ is the maximum

entropy distribution over all distributions with the same domain satisfying $E(\ln(f_0(x))) = \beta$, where β is a given constant. On the other hand, if we have an averaging constraint like $E(g(x)) = \beta$, the maximum entropy density function will have the form $f_0(x) = \exp(-\lambda_0 - \lambda_1\, g(x))$, where λ_0 and λ_1 are constants to normalize the density function and to fit the constraint respectively,

$$\begin{aligned} \int_{-\infty}^{\infty} \exp(-\lambda_0 - \lambda_1\, g(x))\, dx &= 1 \\ \int_{-\infty}^{\infty} g(x)\, \exp(-\lambda_0 - \lambda_1\, g(x))\, dx &= \beta. \end{aligned}$$

One could see that the term λ_0 in the $f_0(x)$ as $\lambda_0 \times 1$ since the normalization constraint can be seen as a constraint with $g(x) = 1$. Therefore, for n averaging constraints, $E(g_i(x)) = \beta_i;\ i = 1, \ldots, n$, one will find that the maximum entropy distribution becomes

$$f_0(x) = \exp\left(-\sum_{i=0}^{n} \lambda_i g_i(x)\right), \tag{4.20}$$

where $g_0(x) = 1$.

If we know n moments values of a random variable, we can generate the maximum entropy distribution of this random variable by setting $g_i(x) = x^i;\ i = 1, \ldots, n$ in the equation (4.20), and calculating the $n+1$ constants from satisfying n moments values and the normalization property,

$$\int_{-\infty}^{\infty} x^i \exp\left(-\sum_{j=0}^{n} \lambda_j x^j\right) = m_i; \quad i = 0, \ldots, n,$$

where $m_0 = 1$. To solve the above set of implicit nonlinear equations, one can use the Taylor series expansion or linearization scheme of the above equations [35],

$$m_i = \bar{m}_i + \sum_{j=0}^{n} \frac{\partial \bar{m}_i}{\partial \lambda_j}\, d\lambda_j; \quad i = 0, \ldots, n,$$

where $\bar{m}_i$ is the i-th moment evaluated from the near-correct set of λ_j. It is clear that $\frac{\partial \bar{m}_i}{\partial \lambda_j} = \bar{m}_{i+j}$, as a result, we have a linear set of algebraic equations

$$\begin{bmatrix} \bar{m}_0 & \bar{m}_1 & \cdots & \bar{m}_n \\ \vdots & & & \bar{m}_{n+1} \\ & & & \vdots \\ \bar{m}_n & \bar{m}_{n+1} & \cdots & \bar{m}_{2n} \end{bmatrix} \begin{bmatrix} d\lambda_0 \\ d\lambda_1 \\ \vdots \\ d\lambda_n \end{bmatrix} = \begin{bmatrix} m_0 - \bar{m}_0 \\ m_1 - \bar{m}_1 \\ \vdots \\ m_n - \bar{m}_n \end{bmatrix}$$

to be solved. From the definition of $\bar{m}_i$, this approach requires us to compute $2n+1$ of one-dimensional integrations everytime we update the values of λ_j. If we have a close initial guess for λ_j then the computation requirement will be minimum. However, there are other requirements for the updating iteration to be convergent. The corresponding implicit nonlinear function should be continuously differentiable, and its Jacobian be nonsingular [149]. Agmon, Alhassid, and Levine [3] proposed a different algorithm for determining the distribution of maximal entropy. Their approach recasted the problem of determining λ_j as a

variational problem.

The maximum entropy method clearly can be extended to solve more than one dimensional problems. As an example, for two dimensional cases, one can write the maximum entropy distribution as follows:

$$f_0(x,y) \;=\; \exp\left(-\sum_{i=0}^{n} \lambda_i \, g_i(x,y)\right).$$

Sobczyk and Trębicki showed the application of two dimensional maximum entropy distribution for approximating the density of nonlinear oscillators [155]. Li and Tankin gave an interesting application of the maximum entropy method for obtaining a prediction of droplet size and velocity distribution in sprays [95]. They used the averaging mass, momentum, and energy conservation relationships as the constraints.

4.4.3 Monte Carlo method

The simplest way to generate the distribution of a random variable when we know its polynomial chaos expansion is by simulating it. From the simulation result we can generate the empirical probability density function. In this case we need to generate Gaussian sampling points because the polynomial chaos expansion uses only Gaussian random variables in its basis functions. Techniques described in chapter 2 can be applied to generate Gaussian sampling points.

We can also use the S language environment [11] or other statistical programming environments to help us generate the empirical density function of a random varible from its polynomial chaos expansion. Two simple examples described in subsection 4.4.1 will be exercised again in here but now with Monte Carlo method in the S language environment.

Example 4.4.3.1

Consider that a random variable z is represented by polynomial chaos expansion as follows:

$$z \;=\; 25.0 + 3.0\,\xi - 1.0\,(\xi^2 - 1) + 0.5\,(\xi^3 - 3\xi).$$

In S language environment, we first generate some Gaussian sampling points, let say 1000 points.

```
> xi <- rnorm(1000)
```

then calculate the value of z for each sampling point,

```
> z <- 25 + 3*xi - (xi^2 - 1) + 0.5*(xi^3 - 3*xi)
```

and finally we can plot the density function of z,

```
> plot(density(z),type="l")
```

Example 4.4.3.2

Similarly, consider that a random variable z is represented by polynomial chaos expansion as follows:

$$z \;=\; 25.0 + 3.0\,\xi_1 + 2.0\,\xi_2.$$

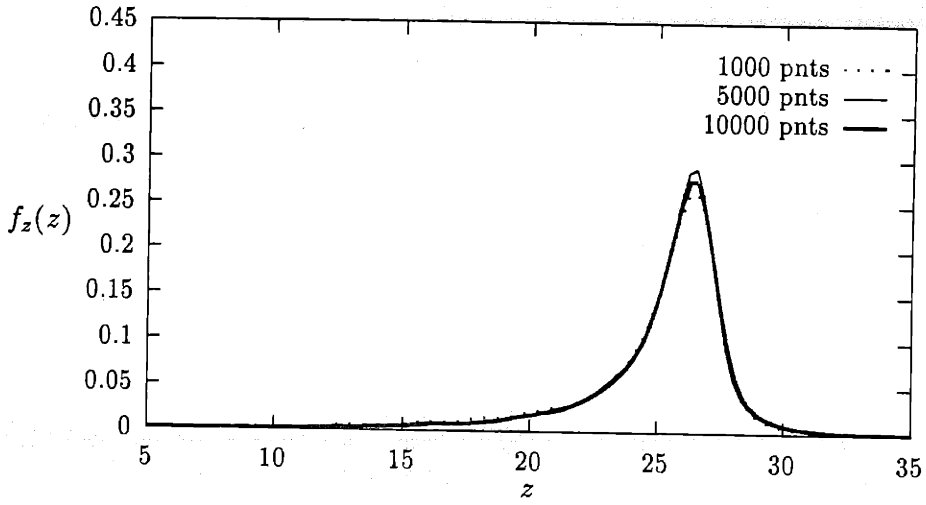


Figure 4-9: First example for the application of Monte Carlo method to generate the probability density function of a polynomial chaos expansion random variable

For this example, we need to generate sampling points from two independent Gaussian random variables,

```
> xi1 <- rnorm(1000)
> xi2 <- rnorm(1000)
```

then calculate the value of z,

```
> z <- 25 + 3*xi1 + 2*xi2
```

and again we can plot the density function of z,

```
> plot(density(z),type="l")
```

As we can see now, the use of Monte Carlo method to generate the empirical density function in S language environment is quite easy and straightforward. However, from the second example we see that when we have multivariate Gaussian random variables as our basis functions, a large number of sampling points is required to produce a good approximation. The use of other previously described methods, such as moments methods, therefore is recommended for cases like that.

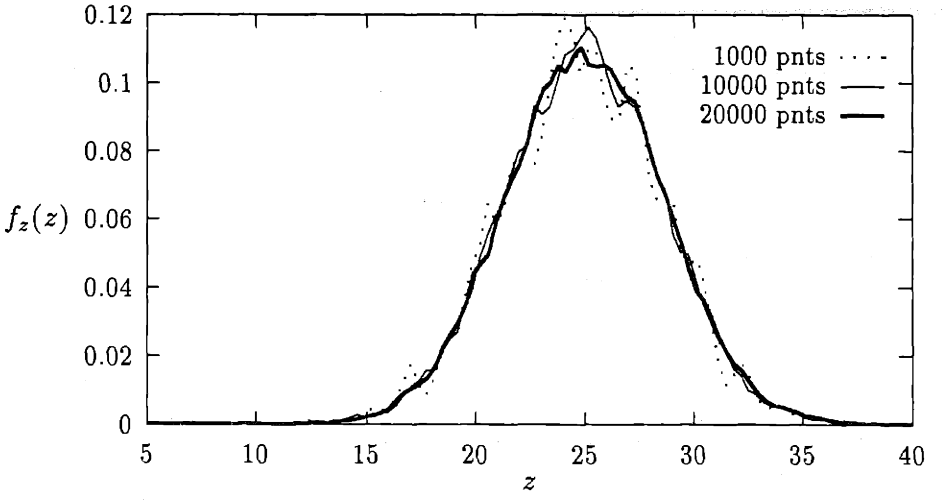


Figure 4-10: Second example for the application of Monte Carlo method to generate the probability density function of a polynomial chaos expansion random variable

Chapter 5

Random Processes or Fields Representation

In the previous chapter, we have discussed general approximations or representations for random variables. For random variables, there are two main classes of representations: the direct and indirect methods. Similarly, we can also apply both types of representations to random processes and random fields. Since a random process or field at a specific value of its parameters can be considered as a random variable, this may suggest that all approaches described in chapter 4 are valid to be used in such cases. However, if one needs to treat the parameters in random processes or fields as a part of variables of interest in the analysis, then one has to consider representations which deal with those parameters directly. This chapter therefore describes such representations, and focuses on a well known member of that class of representations, which is the Karhunen-Loève series expansion.

5.1 Karhunen-Loève series expansion

The notion of Karhunen-Loève series expansion arises not only from the theory of random processes, it also appears in several other fields. As an example, there is the empirical orthogonal function method in meteorology [141]; the proper orthogonal decomposition method is used in operator theory [7]; in statistics, principal component analysis [74] is a widely known tool for data analysis. These methods apply useful properties of Karhunen-Loève series expansions to different applications. We can use principal component analysis to assess the number of degrees of freedom from the data, and to identify principal variables of the data. The empirical orthogonal function and the proper orthogonal decomposition methods can generate basis functions or operators respectively which are empirically or analytically optimal. Similarly, the Karhunen-Loève series expansion may be used to optimally represent random processes.

The Karhunen-Loève series expansion can be categorized as a spectral approach since it decomposes a random process into an infinite series of products of uncorrelated random variables and orthonormal functions of the parameter of the random process. These orthonormal functions have a whole support on the range of values of the parameter. Mathematically, the Karhunen-Loève series expansion of a random process, $x(t,\omega)$, can be written

as follows:

$$x(t,\omega) = \sum_{n=1}^{\infty} c_n \alpha_n(t) \beta_n(\omega),$$

where c_n is the coefficient for the n-th term, and $\alpha_n(t)$ is the n-th orthonormal function defined on the range of parameter values, $t \in (0,T)$. $\beta_n(\omega)$ denotes the n-th uncorrelated random variable which is defined by

$$\int_T x(t,\omega)\ \alpha_n(t)\ dt = c_n \beta_n(\omega). \tag{5.1}$$

Without loss of generality, let us consider that $x(t,\omega)$ is a zero mean random process and its covariance function is denoted by $C(t,s)$. If we multiply equation (5.1) with $x(s,\omega)$, we obtain

$$\int_T x(s,\omega)\ x(t,\omega)\ \alpha_n(t)\ dt = x(s,\omega)\ c_n\ \beta_n(\omega)$$

and by taking the expected value to both sides, we have

$$\int_T C(t,s)\ \alpha_n(t)\ dt = c_n^2\ \alpha_n(s). \tag{5.2}$$

Equation (5.2) represents a homogeneous Fredholm integral equation of the second kind. It means that $\alpha_n(t)$ and c_n^2 can be considered as the n-th eigenfunction and eigenvalue of covariance kernel $C(t,s)$ respectively. Therefore, if we know the covariance function of a random process, theoretically, we can solve equation (5.2) to compute c_n^2 and $\alpha_n(t)$, and then use equation (5.1) to obtain $\beta_n(\omega)$ for $n = 1, \ldots, \infty$.

Since a vector of random variables can be treated as a random process with integer-valued parameter, $t = 1, 2, \ldots, T$, the above procedure is clearly applicable for generating the Karhunen-Loève series expansion of such a vector. The corresponding equations for this random vector can be written as follows:

$$\begin{aligned} \{x_i(\omega)\}_{i=1}^m &= \sum_{j=1}^m \alpha_j\ \beta_j(\omega) \\ \beta_j(\omega) &= \alpha_j^T\ \{x_i(\omega)\}_{i=1}^m \\ \Sigma\ \alpha_j &= \lambda_j\ \alpha_j. \end{aligned}$$

Σ represents the covariance matrix of the random vector, and superscript T denotes a transpose operation. In this scheme, λ_j is not only the j-th eigenvalue, but also the variance of $\beta_j(\omega)$. The value of vector α_j can be obtained by applying singular value decomposition method to the covariance matrix Σ. This technique will be described in the next section. A simple example below shows how we can use the Karhunen-Loève series expansion to represent a correlated random vector and simultaneously reduce its dimensionality.

Example 5.1.1

The most common method for simulating n correlated Gaussian random variables,

$\{x_i(\omega)\}_{i=1}^n$, is the Cholesky decomposition method [138].

$$\{x_i(\omega)\}_{i=1}^n = \{\mu_i\}_{i=1}^n + \mathbf{C}\,\{\xi_j(\omega)\}_{j=1}^n$$

where $\Sigma = \mathbf{C}\,\mathbf{C}^T$ is the covariance matrix of $\{x\}$, $\{\mu_i\}_{i=1}^n$ is the vector of mean values, and $\{\xi_j(\omega)\}_{j=1}^n$ are n independent standard Gaussian random variables.

It is clear that the dimensionality of the problem increases linearly as the number of correlated Gaussian random variables increases. Therefore, when we have, let us say, 25 correlated Gaussian random variables in the system, we need to generate 25 standard independent Gaussian random variables. The question now is, how can we reduce the number of standard independent Gaussian random variables needed to represent those correlated Gaussian random variables?

The objective is to approximate 25 correlated Gaussian random variables with covariance matrix Σ using the minimum number of standard independent Gaussian random variables. The covariance matrix for this problem is given in the following expression.

$$\Sigma_{i,j} = \exp\left(-\frac{R}{24}|i-j|\right); \qquad i,j = 1,\ldots,25$$

where R represents the rate of correlation decay. We will investigate the number of standard independent Gaussian random variables needed to achieve a level of accuracy as a function of the rate of correlation. Therefore, the Mth-order approximation means that only $M \leq 25$ standard independent Gaussian random variables are used for representing those 25 correlated Gaussian random variables.

$$\{x_i(\omega)\}_{i=1}^{25} = \{\mu_i\}_{i=1}^{25} + \sum_{j=1}^{M} \sqrt{\lambda_j}\,\alpha_j\,\xi_j(\omega); \qquad M \leq 25$$

where λ_j and α_j are the j-th empirical eigenvalue and eigenfunction of the covariance matrix Σ respectively from the following relationship

$$\Sigma\,\alpha_j = \lambda_j\,\alpha_j; \qquad j = 1,\ldots,25.$$

For the case M equal to n, or 25 in our example, this type of representation is also called as the canonical expansion of a random vector [130].

When we use only $M \leq 25$ number of standard independent Gaussian random variables, there is an error of approximation. For this example, the L_2-norm error of approximation can be defined as follows:

$$\epsilon_M = \left[\sum_{i,j=1}^{25} \left(\Sigma_{i,j} - \Sigma_{i,j}^{(M)}\right)^2\right]^{\frac{1}{2}}$$

Figure (5-1) shows the error of approximation as a function of the number of standard independent Gaussian random variables used. It also shows that as the rate of correlation decreases (R increases), the number of standard independent Gaussian random variables needed to achieve the same level of accuracy increases. The reason for that comes from a simple analysis that we need to use 25 standard independent Gaussian random variables to represent 25 uncorrelated (independent) Gaussian random variables.

Another measure of the goodness of fit or the level of accuracy is the fractional vari-

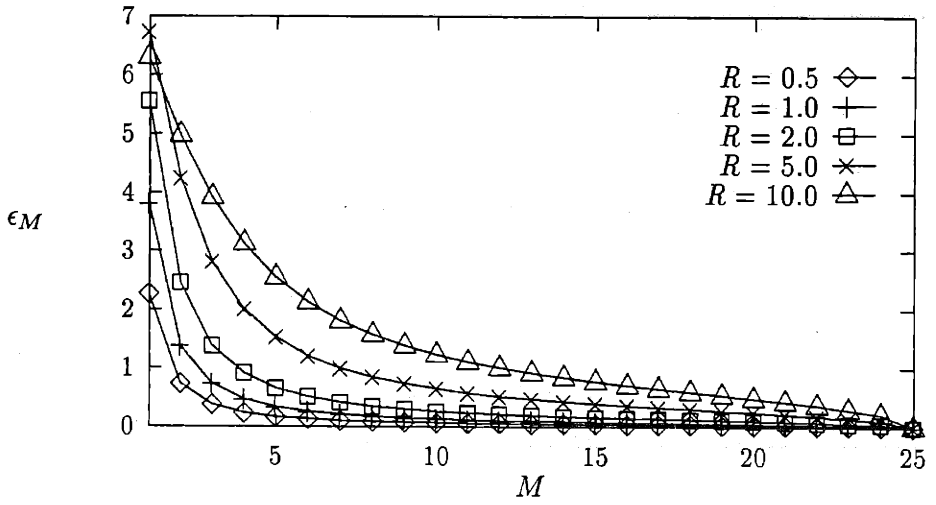


Figure 5-1: The L_2-norm error of approximation for different values of R as a function of number of standard independent Gaussian random variables used

ance [151]. Sometimes this measure is also called the "energy" of approximation,

$$E_M = \frac{1}{E} \sum_{j=1}^{M} \lambda_j$$

where

$$E = \sum_{j=1}^{25} \lambda_j.$$

Figure (5-2) shows the "energy" of approximation as a function of the number of standard independent Gaussian random variables used. For this example, it seems that a practical and sufficient level of "energy" is around 95%. From this figure, we can see that correlated Gaussian random variables with $R = 2.0$ can still be sufficiently approximated by using only a half of the 25 standard independent Gaussian random variables.

5.2 Empirical Karhunen-Loève series expansion

Before we discuss the use of the Karhunen-Loève series expansion to random fields, it is appropriate to describe applications of the Karhunen-Loève series expansion to represent and approximate a deterministic field. Therefore, consider a deterministic field $E(\mathbf{x}, t)$ which is to be approximated. The Karhunen-Loève series expansion of such a field has the form:

$$E(\mathbf{x}, t) = \sum_{n=1}^{N} c_n \alpha_n(t) \beta_n(\mathbf{x}) \tag{5.3}$$

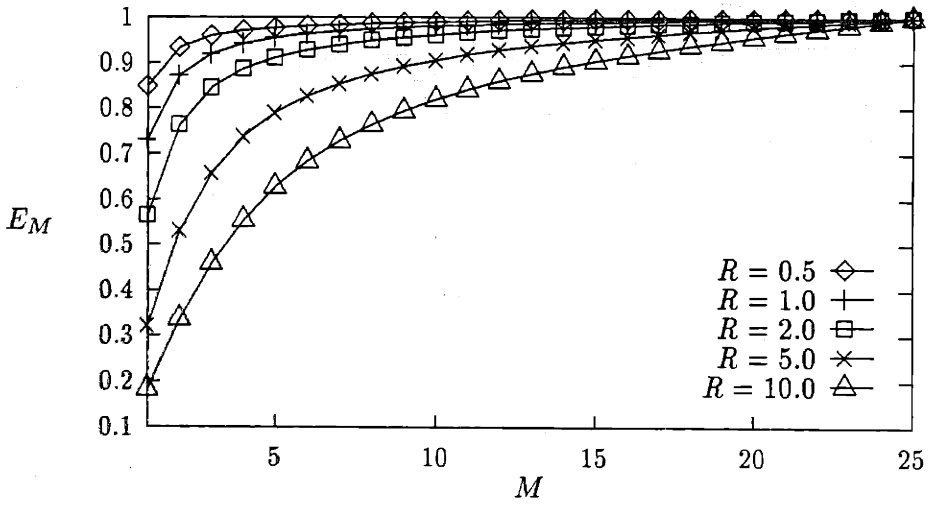


Figure 5-2: The "energy" of approximation for different values of R as a function of number of standard independent Gaussian random variables used

where c_n is the square root of the n-th eigenvalue, $\alpha_n(t)$ is the n-th temporal eigenfunction, and $\beta_n(\mathbf{x})$ is the n-th spatial eigenfunction of the correlation function of $E(\mathbf{x}, t)$. According to Karhunen-Loève series expansion properties, those eigenfunctions should satisfy the normality conditions:

$$\|\alpha_n(t)\|_t^2 = \int_T \alpha_n^2(t)\, dt = 1 \tag{5.4}$$

$$\|\beta_n(\mathbf{x})\|_{\mathbf{x}}^2 = \int_D \beta_n^2(\mathbf{x})\, d\mathbf{x} = 1 \tag{5.5}$$

and also the orthogonality conditions:

$$\int_T E(\mathbf{x}, t)\alpha_n(t)\, dt = c_n\beta_n(\mathbf{x}) \tag{5.6}$$

$$\int_D E(\mathbf{x}, t)\beta_n(\mathbf{x})\, d\mathbf{x} = c_n\alpha_n(t). \tag{5.7}$$

The last two equations are obtained by considering the eigenfunction as an element of a complete set of the orthonormal functions in the $T \times D$ space. From the above equations, the relationship between the eigenvalues and eigenfunctions may be derived as

$$\int_T C(t, s)\alpha_n(s)\, ds = c_n^2\alpha_n(t) \tag{5.8}$$

$$\int_D K(\mathbf{x}, \mathbf{y})\beta_n(\mathbf{y})\, d\mathbf{y} = c_n^2\beta_n(\mathbf{x}) \tag{5.9}$$

where $C(t,s)$ is the correlation matrix of the field in the temporal domain and $K(\mathbf{x},\mathbf{y})$ is the correlation matrix of the field in the spatial domain. They are defined by

$$C(t,s) = \int_D E(\mathbf{x},t)E(\mathbf{x},s)\,d\mathbf{x} \tag{5.10}$$

$$K(\mathbf{x},\mathbf{y}) = \int_T E(\mathbf{x},t)E(\mathbf{y},t)\,dt. \tag{5.11}$$

When the spatial dimension is much larger than the temporal dimension, the first integral equation is clearly preferable to the second one. Assuming the first integral equation is solved, one next needs to obtain the spatial eigenfunctions. The answer to this actually comes from equation (5.6). We can follow a similar technique from Sirovich and Everson [151] who use a snapshot method to calculate the spatial eigenfunctions,

$$\beta_n(\mathbf{x}) = \frac{1}{T}\int_T E(\mathbf{x},t)\alpha_n(t)\,dt. \tag{5.12}$$

However, instead of using equation (5.12), we use equation (5.6) directly to calculate the spatial eigenfunctions. This approach seems more appropriate if one wants to ensure the Karhunen-Loève series expansion properties in (5.6) and (5.7).

The empirical Karhunen-Loève series expansion implies the use of empirical eigenfunctions in the temporal and spatial domains, instead of closed forms. These empirical eigenfunctions are matrices of values of the eigenfunctions at each point in $T \times D$ space, and provide a further advantage: singular value decomposition may be used to obtain the temporal eigenfunctions and eigenvalues and the first integral equation above need not be solved directly. The Karhunen-Loève series expansions can be considered as the orthogonalization of a field, such that the terms in the resulting expansion are uncorrelated, analogous to the singular value decomposition. Thus, the correlation matrix in the temporal domain is decomposed into uncorrelated terms as follows:

$$C(t,s) = \sum_{n=1}^{N} c_n^2 \alpha_n(t)\alpha_n(s). \tag{5.13}$$

The values of c_n^2 and $\alpha_n(t)$ are the diagonal and orthogonal matrices resulting from application of the singular value decomposition method to the correlation matrix:

$$C(t,s) = U \cdot W \cdot V^T \tag{5.14}$$

where $W = \text{diag}\,\{c_n^2\}_{n=1}^N$ and the n-th column of the orthogonal matrix U contains the values of eigenfunction $\alpha_n(t)$ at times t_i, $i = 1,\ldots,N$. Thus, the eigenvalues and temporal eigenfunctions are obtained from orthogonalization of the temporal correlation matrix.

The next step is to calculate the spatial empirical eigenfunctions. Using equation (5.6), we can generate the spatial empirical eigenfunctions,

$$\beta_n(\mathbf{x}) = \frac{1}{c_n}\int_T E(\mathbf{x},t)\alpha_n(t)\,dt \tag{5.15}$$

which are self-normalized,

$$\beta_n(\mathbf{x}) \equiv \frac{\beta_n(\mathbf{x})}{\int_D \beta_n^2(\mathbf{x})\,d\mathbf{x}}. \tag{5.16}$$

These spatial empirical eigenfunctions along with their temporal counterparts and related eigenvalues can now be used as the Karhunen-Loève series expansion of the field.

Now, let us analyze the error of approximation. First we would like to see if the error goes to zero as we increase the number of terms in the expansion. We will use the second norm for evaluating the error since the Karhunen-Loève series expansion produces an approximation with minimum mean-square error (to be discussed later). By definition, the second norm error is

$$\|E(\mathbf{x},t) - E_N(\mathbf{x},t)\|_{t,\mathbf{x}} = \left[\int_D \int_T \left[E(\mathbf{x},t) - \sum_{n=1}^{N} c_n \alpha_n(t)\beta_n(\mathbf{x})\right]^2 dt\, d\mathbf{x}\right]^{\frac{1}{2}} \tag{5.17}$$

where $E_N(\mathbf{x},t)$ denotes the N-term approximation of the field by empirical Karhunen-Loève series expansions. By assuming that

$$E(\mathbf{x},t) = \sum_{n=1}^{\infty} c_n \alpha_n(t)\beta_n(\mathbf{x}) \tag{5.18}$$

we have

$$\|E(\mathbf{x},t) - E_N(\mathbf{x},t)\|_{t,\mathbf{x}} = \left[\sum_{n=N+1}^{\infty} c_n^2\right]^{\frac{1}{2}} \tag{5.19}$$

Theoretically, we can list c_n^2 into a decreasing sequence. It means that as $N \to \infty$ the right hand side of equation (5.19) goes to zero.

The mean-square error of the Karhunen-Loève series expansion approximation of field $E(\mathbf{x},t)$ using N terms is

$$\epsilon_N^2 = \left[\sum_{n=N+1}^{\infty} c_n \alpha_n(t)\beta_n(\mathbf{x})\right]^2. \tag{5.20}$$

The goal now is to minimize the mean-square error in the $T \times D$ space subject to the orthonormality constraints of $\alpha_n(t)$ and $\beta_n(\mathbf{x})$. First, let us look at the minimization with respect to $\beta_n(\mathbf{x})$, and then move to $\alpha_n(t)$. By substituting equation (5.7) into equation (5.20) above and integrating it over the domain we obtain

$$\int_T \int_D \epsilon_N^2\, d\mathbf{x}\, dt = \sum_{n=N+1}^{\infty} \int_D \int_D K(\mathbf{z},\mathbf{y})\beta_n(\mathbf{z})\beta_n(\mathbf{y})\, d\mathbf{z}\, d\mathbf{y} \tag{5.21}$$

with normality constraints as expressed by equation (5.5)

$$\int_D \beta_m(\mathbf{y})\beta_m(\mathbf{y})\, d\mathbf{y} = 1; \quad m = 1,\ldots,\infty. \tag{5.22}$$

For this type of optimization, we can use the Lagrange method to get the stationary condition. Therefore, the Lagrange equation for this objective function and constraints can be written as follows:

$$L = \sum_{n=N+1}^{\infty} \int_D \int_D K(\mathbf{z},\mathbf{y})\beta_n(\mathbf{z})\beta_n(\mathbf{y})\, d\mathbf{z}\, d\mathbf{y} -$$

$$\sum_{m=1}^{\infty} \lambda_m \left[\int_D \beta_m(\mathbf{y}) \beta_m(\mathbf{y}) \, d\mathbf{y} - 1 \right] \tag{5.23}$$

where λ_m is the Lagrange multiplier for the m-th constraint. Differentiating equation (5.23) with respect to $\beta_i(\mathbf{y})$ and setting the result equal to zero gives

$$\int_D \int_D K(\mathbf{z}, \mathbf{y}) \beta_i(\mathbf{z}) \, d\mathbf{z} \, d\mathbf{y} - \lambda_i \int_D \beta_i(\mathbf{y}) \, d\mathbf{y} = 0 \tag{5.24}$$

which is satisfied when

$$\int_D K(\mathbf{z}, \mathbf{y}) \beta_i(\mathbf{z}) \, d\mathbf{z} = \lambda_i \beta_i(\mathbf{y}) \tag{5.25}$$

and it is clearly satisfied by equation (5.9) as a property of the Karhunen-Loève series expansion itself. Similarly for the case of $\alpha_n(t)$, by substituting equation (5.6) into equation (5.20) above and integrating it over the domain we obtain

$$\int_T \int_D \epsilon_N^2 \, d\mathbf{x} \, dt = \sum_{n=N+1}^{\infty} \int_T \int_T C(s,u) \alpha_n(s) \alpha_n(u) \, ds \, du \tag{5.26}$$

with constraints

$$\int_T \alpha_m(s) \alpha_m(s) \, d\mathbf{s} = 1 \quad m = 1, \ldots, \infty. \tag{5.27}$$

Hence, the Lagrange equation for this objective function and constraints becomes

$$\begin{aligned} L \;=\; & \sum_{n=N+1}^{\infty} \int_T \int_T C(s,u) \alpha_n(s) \alpha_n(u) \, ds \, du - \\ & \sum_{m=1}^{\infty} \lambda_m \left[\int_T \alpha_m(s) \alpha_m(s) ds - 1 \right]. \end{aligned} \tag{5.28}$$

Differentiating equation (5.28) with respect to $\alpha_i(s)$ and setting the result equal to zero gives

$$\int_T \int_T C(s,u) \alpha_i(u) \, du \, ds - \lambda_i \int_T \alpha_i(s) \, ds = 0 \tag{5.29}$$

which is satisfied when

$$\int_T C(s,u) \alpha_i(u) \, du = \lambda_i \alpha_i(s) \tag{5.30}$$

and once more it is satisfied by equation (5.8) as a property of Karhunen-Loève series expansion itself. It should be noted that λ_i is c_i^2 in that pair of integral equations (5.8–5.9).

An example to show the application of empirical Karhunen-Loève series expansion for approximating the deterministic field from an inverse problem is given below. This example requires both utilities of the empirical Karhunen-Loève series expansion. We need to reduce the dimensionality of the problem and incorporate all prior information as well. The large dimensionality arises from the discretization of a field, and the prior information of the field comes from the knowledge of the field structure and from observations.

Example 5.2.1

An operator problem in three dimensions is shown in figure (5-3). The operator equation

of this problem can be written as follows:

$$T(x,y,z) = \frac{\sinh \pi\sqrt{2}(1-z)}{\sinh \pi\sqrt{2}}\, T(x,y,0). \tag{5.31}$$

For this example, we will consider two cases. The first case has a smooth input field which is represented by the following expression

$$T_{true}(x,y,0) = \sin \pi x \sin \pi y, \tag{5.32}$$

and the contour plots of this true input field and its first five eigenfunctions can be seen in figure 5-4. On the other hand, the true input field of the second case will not be smooth.

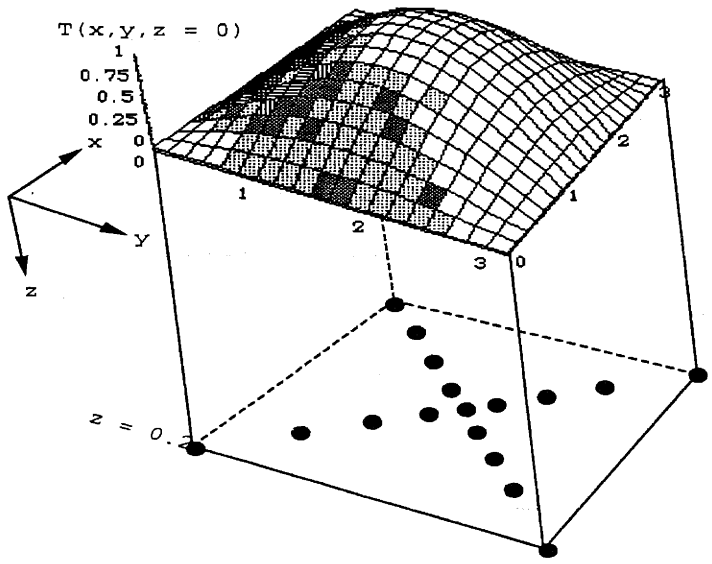


Figure 5-3: Example of nonlinear algebraic operator problem.

Suppose that the input field at $z = 0$ cannot be measured directly or is difficult to obtain accurately. However, we may have limited information on the structure of such a field; for example, it should be symmetric with respect to the middle point, the field should vanish at the boundaries, and we may also have a crude observation of that field. The goal is then to be able to predict the true field by using several accurate measurements or data of $T(x,y,z)$, in this case, at $z = 0.2$.

To approximate this continuous field, we discretize the domain into 20 by 20 grid points. The simplest approach to predict the true discretized field is by using an error minimization strategy which will have dimensionality equal to the number of data points at $z = 0.2$. Since the operator equation is linear with respect to x and y, this approach is limited to prediction at grid points where data is available. Unless the number of such grid points is large, the use of a common interpolation procedure for obtaining values at other grid points may not give a good prediction. Therefore, we need a better scheme which not only captures our prior knowledge and information, but can also reduce the dimensionality of the minimization problem.

Since obtaining accurate data at a specific grid point may be expensive, we need to

minimize the number of measurements. In practice, one may do a measurement design, or for this example, our limited information on the structure of the input field suggests that we use symmetric data points. Therefore, we choose 17 data points as shown in figure (5-3).

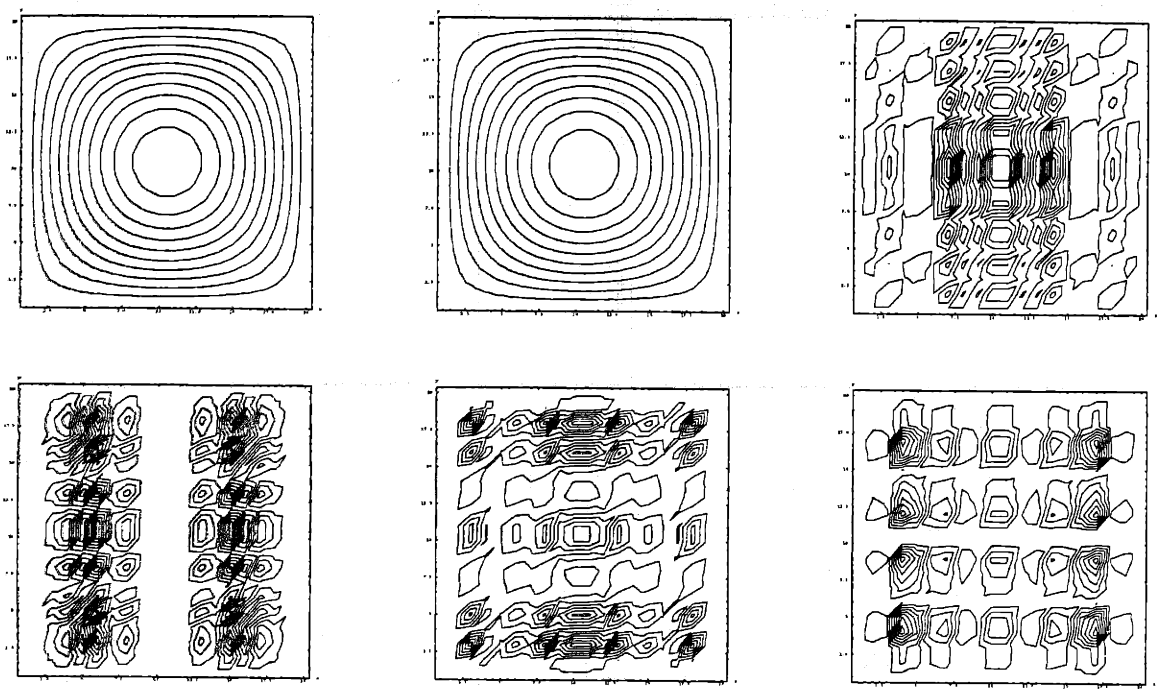


Figure 5-4: True input field and its first five eigenfunctions for the first case

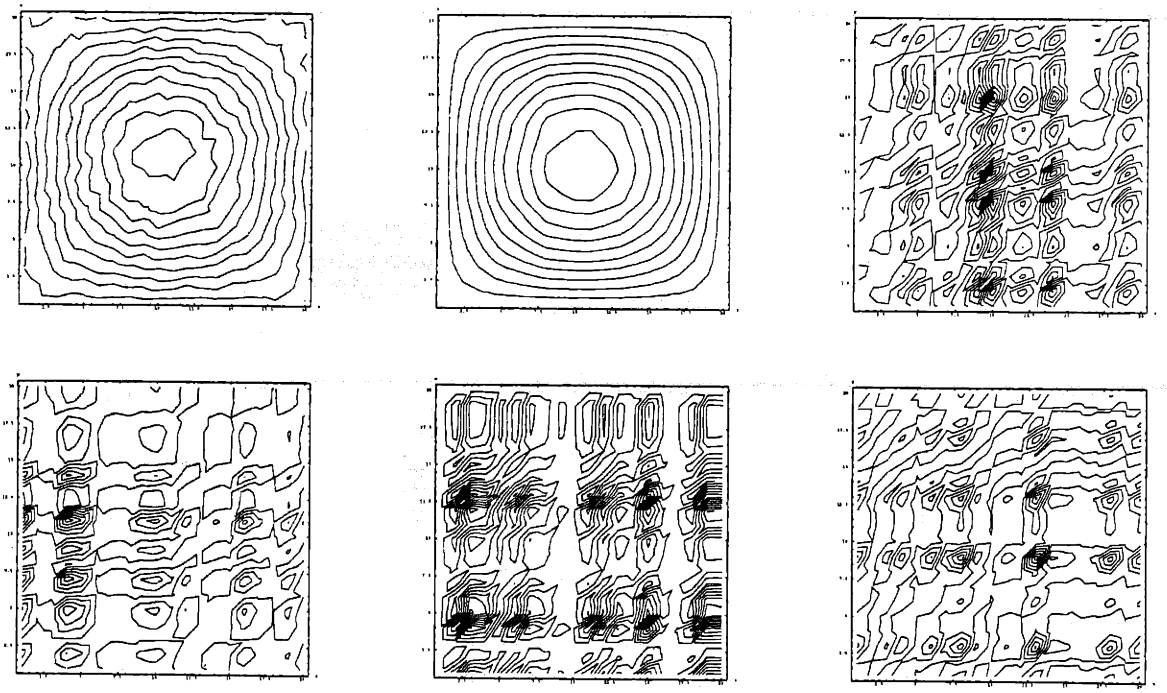


Figure 5-5: Prior or observed input field (obtained by randomly perturbed the true input field with noise to signal ratio around 0.1) and its first five eigenfunctions for the first case

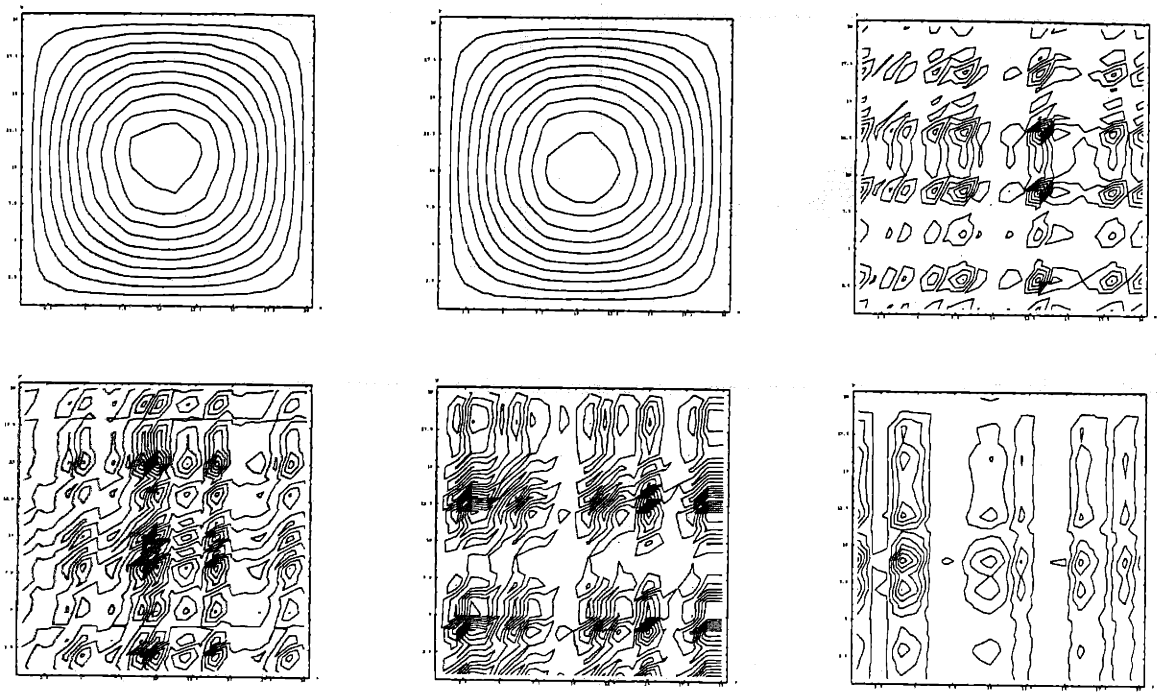


Figure 5-6: Optimized input field (using 17 points at z = 0.2 and BFGS variant of DFP minimization method to get optimal coefficients) and its first five eigenfunctions for the first case

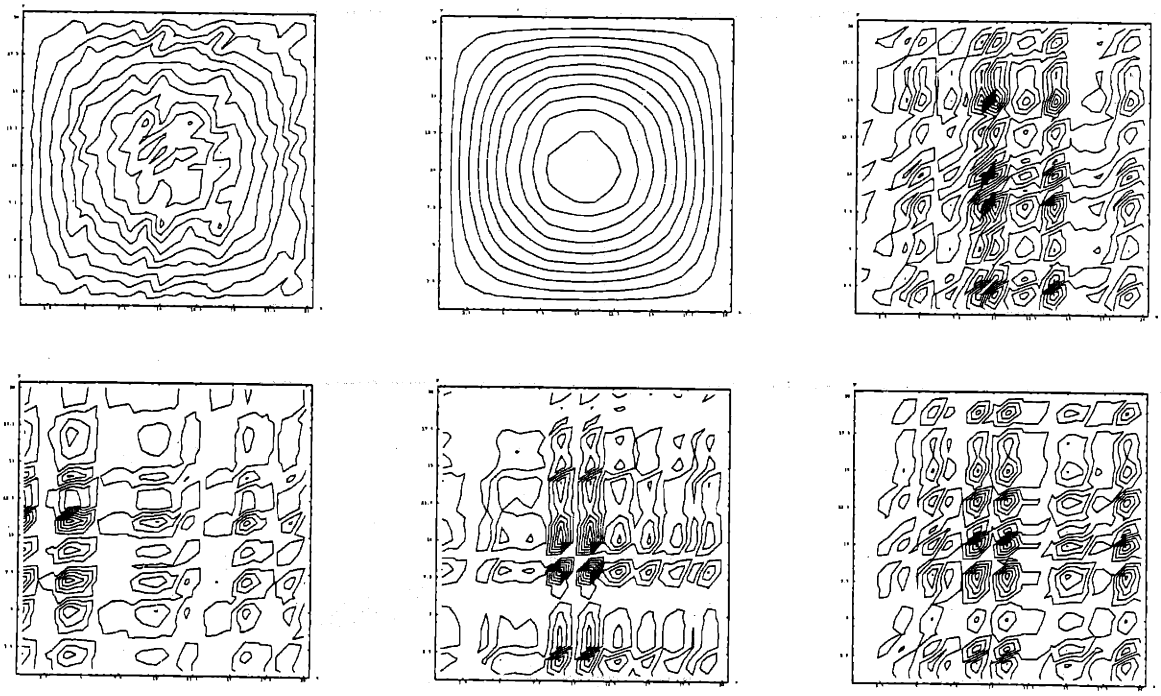


Figure 5-7: True input field and its first five eigenfunctions for the second case

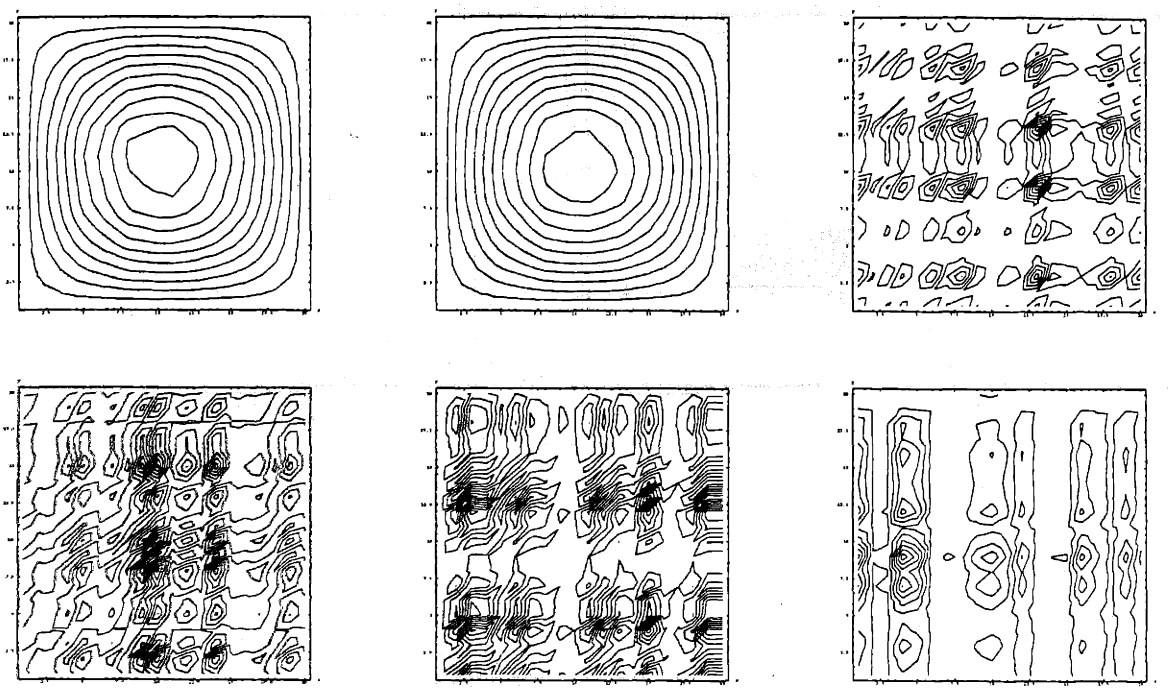


Figure 5-8: Prior of observed input field and its first five eigenfunctions for the second case

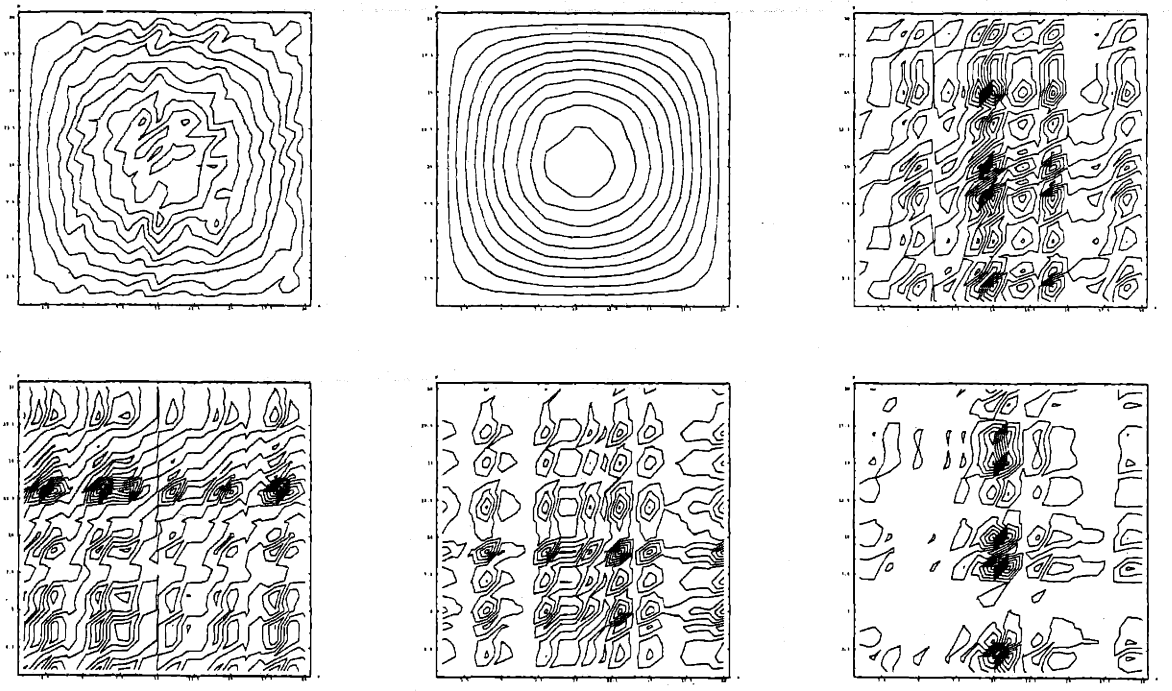


Figure 5-9: Optimized input field (using 17 points at z = 0.2 and BFGS variant of DFP minimization method to get optimal coefficients) and its first five eigenfunctions for the second case

For the first case, we shall assume that we have a crude observation of the input field. The observed field contains noise which is around 10% of signal. The objective is then to reduce the noise of the observed input field on 400 grid points by only measuring accurately 17 grid points at $z = 0.2$. Since our prior knowledge of the input field is in the observed field, we apply the empirical Karhunen-Loève series expansion approach to obtain the coherent structures of this observed field. In this case, we use **EISPACK** subroutines [50] to solve the corresponding singular value decomposition problems. These coherent structures become our prior information for generating the optimized input field. Figure (5-5) shows the contour plots of the observed field and its first five empirical eigenfunctions or coherent structures.

We incorporate our prior knowledge or information in terms of the first five empirical eigenfunctions and the operator model into the optimization scheme. For our example, we choose the L_2-norm of error as our objective function.

$$\min_{\{c_n\}_{n=1}^{5}} \quad \left[\sum_{i=1}^{17} (T_{data}(x_i, y_i, 0.2) - T_{calc}(x_i, y_i, 0.2))^2\right]^{\frac{1}{2}} \tag{5.33}$$

where

$$T_{calc}(x_i, y_i, 0.2) \;=\; \frac{\sinh \pi\sqrt{2}(1 - 0.2)}{\sinh \pi\sqrt{2}}\, T_{prior}(x_i, y_i, 0)$$

and

$$T_{prior}(x_i, y_i, 0) \;=\; \sum_{n=1}^{5} c_n \alpha_n(x_i)\beta_n(y_i),$$

and $\alpha_n(x)\beta_n(y)$ represents the n-th empirical eigenfunction from the observed input field. The optimization scheme yields a set of optimal coefficients, $\{c_n\}_{n=1}^{5}$, which in turn can be used to generate the posterior or optimized input field.

$$T_{posterior}(x, y, 0) \;=\; \sum_{n=1}^{5} c_n \alpha_n(x)\beta_n(y).$$

It is clear from equation (5.33) that the dimensionality of the optimization depends only on the number of coherent structures we use as our prior information. For this example, the dimensionality of optimization is 5, and we use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) variant of Davidon-Fletcher-Powell (DFP) minimization method to obtain the optimal coefficients [127].

The posterior input field for the first case can be seen in figure (5-6). As we can see from this figure, the optimization scheme acts as a filter to the noise. This result could also be obtained by using the Fourier expansion and retaining only the first few coefficients. Therefore, it would be of interest to see whether the empirical Karhunen-Loève expansion could be used in the optimization of a smooth prior input field when the true input field is actually not smooth.

Figure (5-7) and (5-8) show the contour plots, for the second case, of the true input field and the prior information of input field and their first five empirical eigenfunctions respectively. We see that the prior input field is smoother than the true input field. For this case, the use of Fourier expansion or other filtering techniques will not give us a better

input field than the prior one. Therefore, we use the similar procedure as in the first case, and the contour plots of the optimized input field can be seen in figure (5-9). We notice that, in the second case, the second empirical eigenfunction of the optimized field is almost the same as that of the true input field. This means that the second empirical eigenfunction is important for defining the field.

It should be noted that the optimization procedure can be done iteratively in order to update the prior input field. Therefore, the posterior input field at the j-th cycle becomes the prior input field for the $j+1$-th cycle. Figures (5-9) and (5-6) are actually obtained after conducting 3-cycle optimizations.

In general, we can use the empirical Karhunen-Loève series expansion to accurately approximate any field. Since the complete set of eigenfunctions from that expansion spans the L^2 space [7], an arbitrary regular field in L^2 space can then be theoretically approximated using the same set of eigenfunctions. In other words, in practice, chemical or physical phenomena with similar underlying structure may be represented by the same set of empirical eigenfunctions [6, 84, 86, 137, 175]. Based on this fact, we can incorporate our prior knowledge of a field and use it to generate a better prediction.

Retaining only the coherent structures with higher energy as our prior information in the optimization scheme suggests that we put more weight in keeping the structure of the field. Similarly, we could also put more weight to a specific coherent structure by adding constraints to the optimization problem; for example, the coefficient associated with the first empirical eigenfunction should be greater than others. Other constraints which describe the limit of physical or chemical phenomena could also be added to the problem. As an example, if we deal with concentration of species then we can specify that the posterior field of concentration will be greater than zero for all grid points.

This example shows that the empirical Karhunen-Loève series expansion can help us in reducing the dimensionality of the problem, or transforming the original dimensionality to another measure of dimensionality. Since truncating the exact representation introduces an error of approximation, a sensitivity analysis of the result to the number of coherent structures used should be conducted.

5.3 Random fields representations

From previous discussions, it is clear that the Karhunen-Loève series expansion is applicable to represent a random field. However, the covariance function now has two or more variables depending on the dimensionality of the field. As an example, the corresponding Fredholm integral equation for a two dimensional random field becomes

$$\int_T \int_U C\left(t_1, t_2; u_1, u_2\right)\ \alpha_n\left(t_2, u_2\right)\ dt_2\ du_2 \quad = \quad \lambda_n\ \alpha_n\left(t_1, u_1\right), \tag{5.34}$$

and the uncorrelated random variables element for this case can be obtained from the following equation:

$$\beta_n(\omega) \quad = \quad \frac{1}{\lambda_n} \int_T \int_U x(t, u, \omega)\ \alpha_n\left(t, u\right)\ dt\ du.$$

Consequently, the random field is represented in the form

$$x(t,u,\omega) \;=\; \sum_{n=1}^{\infty} \sqrt{\lambda_n}\; \alpha_n(t,u)\; \beta_n(\omega).$$

To solve the above Fredholm integral equation, we can use a Galerkin type procedure [56]. The main idea is to approximate the eigenfunctions by the use of a complete set of functions in the Hilbert space. It means that each eigenfunction of the covariance kernel may be written as

$$\alpha_n(\mathbf{t}) \;=\; \sum_{i=1}^{M} d_{i,n}\; h_i(\mathbf{t}),$$

where, to simplify the notation, $\mathbf{t}$ denotes a vector of parameters of random field such as spatial and temporal coordinates, and $h_i(\cdot)$ is a member of the complete set of functions. It should be noted that the above representation for eigenfunctions contains an error resulting from truncating the summation after the M-th term. Therefore, the induced error from this approximation to the Fredholm integral equation becomes

$$\epsilon_M \;=\; \sum_{i=1}^{M} d_{i,n} \left[\int_{\mathbf{T}} C(\mathbf{t_1},\mathbf{t_2})\; h_i(\mathbf{t_2})\; d\mathbf{t_2} \;-\; \lambda_n\; h_i(\mathbf{t_1}) \right].$$

In the spirit of the Galerkin approach, this error should be made orthogonal to the approximating space. Such an orthogonalization process yields the following generalized algebraic eigenvalue problem,

$$\mathbf{C}\,\mathbf{D} \;=\; \Lambda\,\mathbf{B}\,\mathbf{D},$$

where $\mathbf{C}$, $\mathbf{B}$, Λ, and $\mathbf{D}$ are four M dimensional matrices whose elements are given as follows

$$\begin{aligned}
\mathbf{C}_{i,j} &= \int_{\mathbf{T}} \int_{\mathbf{T}} C(\mathbf{t_1},\mathbf{t_2})\; h_i(\mathbf{t_2})\; h_j(\mathbf{t_1})\; d\mathbf{t_2}\; d\mathbf{t_1}, \\
\mathbf{B}_{i,j} &= \int_{\mathbf{T}} h_i(\mathbf{t})\; h_j(\mathbf{t})\; d\mathbf{t}, \\
\mathbf{D}_{i,j} &= d_{i,j}, \\
\Lambda_{i,j} &= \delta_{ij}\; \lambda_i.
\end{aligned}$$

It is clear that we need a $2 \times \dim(\mathbf{T})$ (where $\dim(\mathbf{T})$ denotes the dimension of the vector $\mathbf{T}$) dimensional integration subroutine to generate this eigenvalue problem. It is of interest, therefore, to see whether we can apply a methodology described in the previous section to further decomposing these multi-dimensional eigenfunctions.

In a certain condition, we may actually transform the above multi-dimensional problem into several one-dimensional problems. As an example, let us consider a two-dimensional random field. If the covariance function is separable

$$C(t_1,t_2;u_1,u_2) \;=\; C_t(t_1,t_2)\; C_u(u_1,u_2),$$

the eigenfunction can have the following form

$$\alpha_n(t,u) \;=\; \alpha_{t,n}(t)\; \alpha_{u,n}(u),$$

and similarly, the eigenvalue can be written as

$$\lambda_n \;=\; \lambda_{t,n}\,\lambda_{u,n}.$$

As a result, equation (5.34) becomes two separate eigenvalue problems

$$\begin{aligned}\int_T C_t(t_1,t_2)\;\alpha_{t,n}(t_2)\;dt_2 &= \lambda_{t,n}\,\alpha_{t,n}(t_1),\\ \int_U C_u(u_1,u_2)\;\alpha_{u,n}(u_2)\;du_2 &= \lambda_{u,n}\,\alpha_{u,n}(u_1).\end{aligned}$$

These two eigenvalue problems then can be solved separately using standard numerical subroutines or techniques described in previous sections.

Chapter 6

Stochastic Inequality Models

Many problems in chemical and environmental engineering require the use of inequality models or constraints. For example, product specifications, or the capability of process equipments are best described by inequalities. Similarly, threshold values and risk levels [65] are natural instances of inequality models. For some problems such as flexibility index calculations [163], the inequality model can be solved separately, but for many others, it is coupled to the equality models, or becomes a part of the constraints in an optimization model.

To conduct uncertainty analyses for a chemical or environmental engineering system, there is consequently a need for a systematic methodology for dealing with stochastic inequality models or constraints. There are two main methods available for solving stochastic inequality models: the recourse technique and the chance-constrained method. Both methods have been developed within the field of stochastic programming to handle stochastic inequality constraints in the optimization model [30, 189]. The recourse technique will be described in the first section and the chance-constrained method in the second section. In the last section, the use of both techniques to solve stochastic programming problems is briefly reviewed.

6.1 Recourse technique

A stochastic inequality model can generally be written as

$$\mathbf{G}(\mathbf{z}; \mathbf{d}, \theta) \leq \mathbf{0},$$

where $\mathbf{z}$ is the vector of control variables in the model respectively, $\mathbf{d}$ is the vector of deterministic parameters, θ denotes the vector of stochastic or uncertain parameters, and $\mathbf{G}(\cdot)$ represents a vector of functions of $\mathbf{z}$. If this inequality model is a part of the constraints in an optimization model, $\mathbf{d}$ can be considered as the vector of design or decision variables. Furthermore, if this inequality model is coupled with an equality model, then the vector of state variables $\mathbf{x}$ can also be incorporated in the inequality formulation.

For given values of the vectors $\mathbf{d}$ and $\mathbf{z}$, the varying argument left in $\mathbf{G}$ is the uncertain parameters θ,

$$\mathbf{G}(\theta) \leq \mathbf{0}.$$

It means that this set of inequalities may or may not be satisfied for a certain value of θ. The

measure of this level of satisfaction actually becomes the point where the differences between recourse and chance-constrained methods arise. The recourse technique measures the extent of violation, but the chance-constrained method controls the probability or frequency of violation.

In order to measure the extent of violation of a set of inequalities, recourse technique decomposes the slack variables of these inequalities into pairs. The inequality model becomes

$$\begin{aligned} \mathbf{G}(\theta) + \mathbf{q}^+ - \mathbf{q}^- &= \mathbf{0} \\ \mathbf{q}^+ &\geq \mathbf{0} \\ \mathbf{q}^- &\geq \mathbf{0}, \end{aligned} \tag{6.1}$$

where $\mathbf{q}^+$ and $\mathbf{q}^-$ are vectors of such pairs. Since for different realizations of θ, vectors $\mathbf{q}^+$ and $\mathbf{q}^-$ may have different values, those two vectors can then be considered as vectors of random variables.

Because the corresponding elements of vectors $\mathbf{q}^+$ and $\mathbf{q}^-$ are complementary to each other, which means that if $q_i^+ \neq 0$ then $q_i^- = 0$ and vice-versa, we can generate the probability density function for each element of both vectors by simultaneously simulating the uncertain parameters θ and solving equations (6.1) and (6.2)

$$q_i^+ \, q_i^- = 0; \quad \forall i. \tag{6.2}$$

In other words, the formulation of a stochastic inequality model in the form of equation (6.1) gives us a control of its degree of violation, such as the mean value of violation, $\mu(q_i^-)$. This formulation also provides the probability of satisfying inequalities which can be computed as follows

$$P(\mathbf{G}(\theta) \leq 0) = P(\mathbf{q}^- - \mathbf{q}^+ \leq \mathbf{0}),$$

where $P(\cdot)$ denotes the probability function.

The application of the recourse technique to compute the mean value of violation and the probability for satisfying a set of inequalities will be given below. This example illustrates not only the use of the recourse technique, but also the use of the Monte Carlo method for computing the stochastic flexibility index from a given set of active reduced constraints. It should be noted that the stochastic flexibility index is the probability of the projection of $\mathbf{G}(\mathbf{z}; \mathbf{d}, \theta) \leq \mathbf{0}$ in the θ space at a specific value of vector $\mathbf{d}$.

Example 6.1.1

The following problem is taken from Grossmann and Floudas [59]. In this example, a simple system of liquid transportation is considered and shown in figure 6-1. The data set of the parameters in the problem are given in table 6.1, and the uncertain parameters are assumed to be distributed as independent Gaussian distributed random variables.

A set of active reduced constraints can be generated for such a system as follows (please see [59] for a complete formulation of the model and a detailed description of the parame-

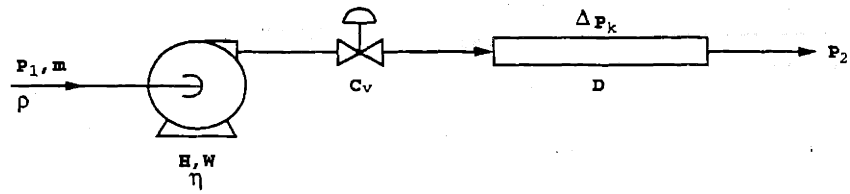


Figure 6-1: Configuration of pump and pipe run.

Parameter	Mean value	Standard deviation
H	1.3 kJ/kg	0.0 kJ/kg
D	0.0762 m	0.0 m
C_v^{max}	0.039673	0.0
r	0.05	0.0
ϵ	20.0 kPa	0.0 kPa
P_1	100.0 kPa	0.0 kPa
ρ	1000.0 kg/m^3	50.0 kg/m^3
m	10.0 kg/s	2.0 kg/s
k	9.101 10^{-6} (kPa)(kg/s)$^{-1.84}$(m$^{5.16}$)	0.45505 10^{-6} (kPa)(kg/s)$^{-1.84}$(m$^{5.16}$)
P_2'	800.0 kPa	500.0 kPa

Table 6.1: Mean values and standard deviations of the parameters in example of pump and pipe run.

ters):

$$h_1 \equiv P_1 + \rho H - \epsilon - \frac{m^2}{\rho r^2 C_v^{max2}} - km^{1.84} D^{-5.16} - P_2' \leq 0 \tag{6.3}$$

$$h_2 \equiv -P_1 - \rho H - \epsilon + \frac{m^2}{\rho C_v^{max2}} + km^{1.84} D^{-5.16} + P_2' \leq 0 \tag{6.4}$$

$$h_3 \equiv -\epsilon \leq 0 \tag{6.5}$$

$$h_4 \equiv -0.5 C_v^{max}(1-r) \leq 0 \tag{6.6}$$

For this example, we will use the Monte Carlo method with Hammersley-Wozniakowski sampling points to calculate the mean value and standard deviation of violations and the probability of satisfying them. To transform those uniformly distributed Hammersley-Wozniakowski sampling points to the corresponding normal variates sampling points, we use the Box-Müller method. The application of the Monte Carlo method is relatively simple for this example, and its procedure is given as follows:

1. Set $p \leftarrow 0$

2. Generate an appropriately distributed sampling point, in this case four standard independent normal variates, ξ_1, ξ_2, ξ_3, and ξ_4, and then compute the corresponding values of uncertain parameters

$$\rho = 1000.0 + 50.0\ \xi_1$$

$$
\begin{aligned}
m &= 10.0 + 2.0\,\xi_2 \\
k &= 9.101\ 10^{-6} + 0.45505\ 10^{-6}\,\xi_3 \\
P_2' &= 800.0 + 500.0\,\xi_4
\end{aligned}
$$

3. Calculate the value of h_1, h_2, h_3, and h_4
4. For $i = 1, \ldots, 4$, If $h_i \leq 0$ then $q_i^+ = -h_i$ and $q_i^- = 0$ otherwise $q_i^+ = 0$ and $q_i^- = h_i$.
5. If $\forall i,\ q_i^- - q_i^+ \leq 0$, then $p \leftarrow p + 1$.
6. If number of sampling points used is less than the desired number, N, then go back to the second step, otherwise evaluate the next step.
7. Perform statistical analysis

$$
\begin{aligned}
\mu(q_i^-) &= \frac{1}{N} \sum_{j=1}^{N} q_{i,j}^-; \quad i = 1, \ldots, 4 \\
\sigma(q_i^-) &= \sqrt{\frac{1}{N-1} \sum_{j=1}^{N} \left(q_{i,j}^- - \mu(q_i^-)\right)^2}\ ; \quad i = 1, \ldots, 4 \\
P(\mathbf{h} \leq \mathbf{0}) &= \frac{p}{N}
\end{aligned}
$$

Table 6.2 shows the results of this example. Since the third and fourth constraints, h_3, h_4, do not involve uncertain parameters, variables q_3^-, q_3^+, q_4^-, and q_4^+ are not random variables. It means that the standard deviation or variance of those variables should equal to zero. It should however be pointed out that the zero variance results of q_1^- do not imply that q_1^- is a deterministic variable. It is possible that for a certain combination of the values of uncertain parameters, the value of q_1^- may not be deterministically equal to zero, but with these choices of sampling points, the Monte Carlo method unfortunately has not found such a case.

From the notion of stochastic flexibility for a system [163], the value of $P(\mathbf{h} \leq \mathbf{0})$ is known as the stochastic flexibility index, and it is the probability of satisfying all inequality constraints simultaneously. For this example, the index resulted from simulating 10000 points therefore equals to $1 - 0.37 = 0.63$.

The simulation results can also give us the empirical density functions of q_1^+, q_2^+, and q_2^- which are shown in figures 6-2 – 6-4. They are produced by applying the standard density plot routine from the **S** language [11].

The above example shows that the original formulation of the recourse technique which is given in equation (6.1) cannot directly control the probability of satisfying the inequality constraints. However, this limitation can be eliminated by adding the following constraint which involves the pairs of slack variables,

$$
P(\mathbf{q}^- - \mathbf{q}^+ \leq \mathbf{0};) \ \geq \ \alpha,
$$

where α is a number between zero to one, and it measures the control for satisfying inequality constraints.

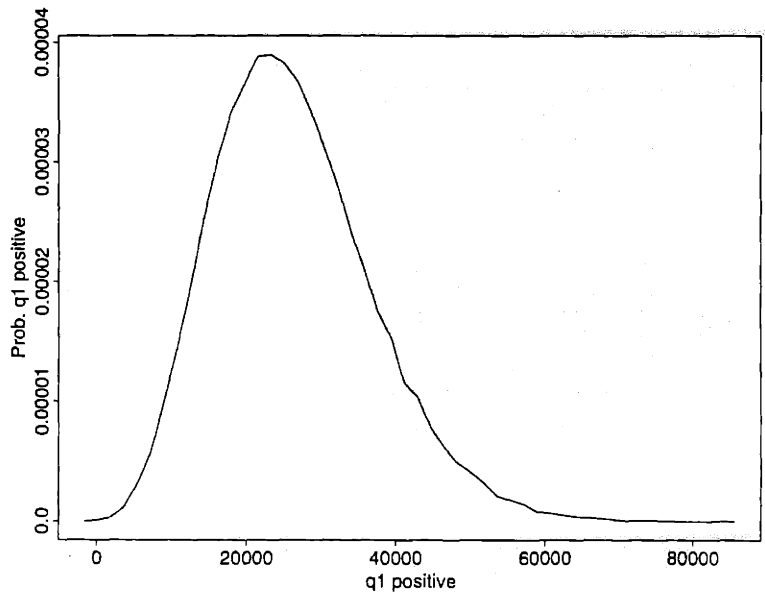


Figure 6-2: Probability density function of q_1^+.

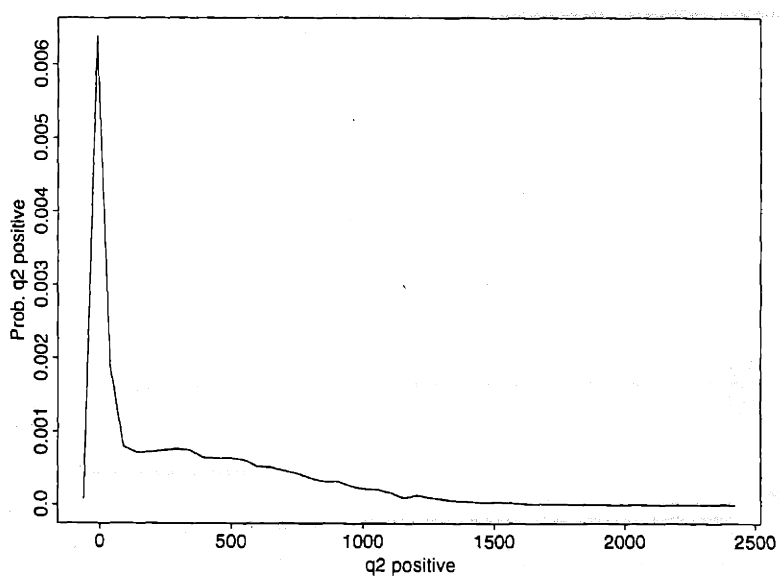


Figure 6-3: Probability density function of q_2^+.

Measures	1000 points	5000 points	10000 points
$\mu(q_1^+)$	26278.30656	26294.45329	26296.72843
$\sigma(q_1^+)$	10507.43636	10537.72218	10540.55719
$\mu(q_1^-)$	0	0	0
$\sigma(q_1^-)$	0	0	0
$\mu(q_2^+)$	308.29251	308.84218	308.91525
$\sigma(q_2^+)$	363.34143	365.01152	365.26011
$\mu(q_2^-)$	135.66150	136.56070	136.60177
$\sigma(q_2^-)$	249.21010	251.05111	251.21835
$\mu(q_3^+)$	20.0	20.0	20.0
$\sigma(q_3^+)$	0	0	0
$\mu(q_3^-)$	0	0	0
$\sigma(q_3^-)$	0	0	0
$\mu(q_4^+)$	0.01884	0.01884	0.01884
$\sigma(q_4^+)$	0	0	0
$\mu(q_4^-)$	0	0	0
$\sigma(q_4^-)$	0	0	0
Prob. violation	0.3620	0.3708	0.37

Table 6.2: Results of pump and pipe run example.

6.2 Chance-constrained method

The use of pairs of slack random variables in fact becomes unnecessary if we only want to control the frequency of violating a set of inequalities, because we can directly formulate the probabilistic inequality as follows:

$$P(\mathbf{G}(\mathbf{z};\mathbf{d},\theta) \leq \mathbf{0}) \quad \geq \quad \alpha.$$

For the case of a set of active reduced constraints [59], such as the example of the pump and pipe run shown previously, the only variables in the active reduced constraints are the design parameters, $\mathbf{d}$, and uncertain parameters, θ,

$$P(\mathbf{G}(\mathbf{d},\theta) \leq \mathbf{0}) \quad \geq \quad \alpha.$$

It means that we can change the values of design parameters accordingly to satisfy the above probabilistic inequalities. While for given values of design parameters, we can not control but only calculate the value of $P(\mathbf{G}(\theta) \leq \mathbf{0})$, Monte Carlo methods can clearly be applied to compute that probability value, and to assess whether the probabilistic inequalities are satisfied or not.

One may also use the analytical approach when the closed form of $P(\mathbf{G}(\mathbf{d},\theta) \leq \mathbf{0})$ is available. As an example, if all slack variables in the inequalities, $\mathbf{h} \equiv \mathbf{G}(\mathbf{d},\theta)$, can be assumed to be independently normal distributed,

$$f_{\mathbf{h}}(\mathbf{h}) \quad = \quad \prod_{i=1}^{N} \frac{1}{\sigma(h_i)\,\sqrt{2\pi}} \exp\left(-\frac{(h_i - \mu(h_i))^2}{2\,\sigma^2(h_i)}\right),$$

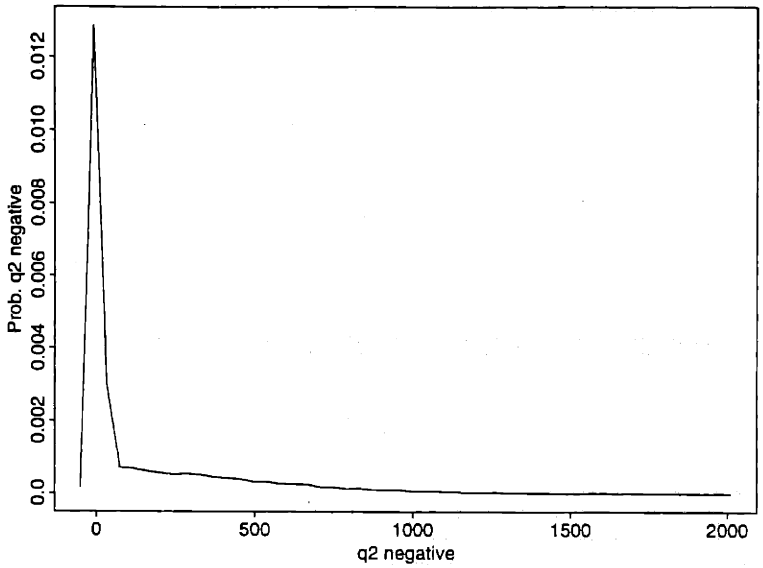


Figure 6-4: Probability density function of q_2^-.

where N is the number of inequalities, and $\mu(h_i)$ and $\sigma(h_i)$ represent the mean value and standard deviation of h_i respectively, the cumulative density function for this vector $\mathbf{h}$ becomes

$$F_{\mathbf{h}}(\mathbf{h}) \;=\; \prod_{i=1}^{N} \frac{1 + \mathrm{Erf}\left(\frac{h_i - \mu(h_i)}{\sigma(h_i)\sqrt{2}}\right)}{2},$$

and the probabilistic inequality can be written as follows:

$$\prod_{i=1}^{N} \frac{1 + \mathrm{Erf}\left(\frac{h_i - \mu(h_i)}{\sigma(h_i)\sqrt{2}}\right)}{2} \;\geq\; \alpha.$$

In some cases, the normality and independence assumptions might be sufficient for producing a rough approximation of the solutions. Furthermore, when it is difficult to assign a representative density function for each uncertain parameter, such an approximation approach may be justifiable.

An obvious application for this chance-constrained formulation is the assessment of the stochastic flexibility index for a given system. As an example for assessing the stochastic flexibility index in the design phase, the previous problem of the pump and pipe run is considered again. The problem then consists of choosing from several design alternatives for which the stochastic flexibility index of the chosen systems should be at least equal to 0.95,

$$P(\mathbf{h} \;\leq\; \mathbf{0}) \;\geq\; 0.95.$$

Example 6.2.1

Several design alternatives for the pump and pipe run problem are given in table 6.3 below. Their values are obtained from preliminary design calculations. At this point, we need to reduce the number of design alternatives based on the consideration of their

stochastic flexibility indices. To compute the stochastic flexibility index for each design

Design Variables	A_1	A_2	A_3	A_4
H	1.3	1.3	1.7	1.7
D	0.1016	0.0762	0.1016	0.127
C_v^{max}	0.04	0.06	0.05	0.04

Table 6.3: Design variables of example problem.

alternative, we will use the Monte Carlo method with 5000 Hammersley-Wozniakowski sampling points. We apply the same sampling points to each design alternative to inherently reduce the variance of comparison (see chapter 2). The following procedure illustrates the use of the Monte Carlo method for computing stochastic flexibility index of the pump and pipe run problem:

1. Set $p \leftarrow 0$
2. Generate an appropriately distributed sampling point, in this case four standard independent normal variates, ξ_1, ξ_2, ξ_3, and ξ_4, and then compute the corresponding values of uncertain parameters

$$\begin{aligned} \rho &= 1000.0 + 50.0\ \xi_1 \\ m &= 10.0 + 2.0\ \xi_2 \\ k &= 9.101\ 10^{-6} + 0.45505\ 10^{-6}\ \xi_3 \\ P_2' &= 800.0 + 500.0\ \xi_4 \end{aligned}$$

3. Calculate the value of h_1, h_2, h_3, and h_4
4. If $\forall i, h_i \leq 0$, then $p \leftarrow p + 1$.
5. If number of sampling points used is less than the desired number, N, then go back to the second step, otherwise evaluate the next step.
6. Perform statistical analysis

$$P(\mathbf{h} \leq \mathbf{0}) = \frac{p}{N}$$

From applying this procedure to each design alternative, we can calculate the stochastic flexibility indices for alternatives A_1, A_2, A_3, and A_4, and they are $0.8198, 0.6552, 0.9620$, and 0.9666 respectively. These results show that alternatives A_1 and A_2 clearly do not satisfy the flexibility requirement. We are then left with only two design alternatives, and the choice between alternatives A_3 and A_4 will depend on other measures, such as cost or a safety factor.

To simultaneously optimize design alternatives with several measures such as stochastic flexibility index and cost factor, the stochastic inequality model should be treated as a part of an optimization model. These stochastic inequalities may represent the specifications, limits or capabilities, ranges of interest, and performances of the system in the optimization model. Depending on the main objective of the optimization model, one can use either one or both recourse and chance-constrained representations for stochastic inequality constraints.

6.3 Stochastic inequality constraints

As mentioned previously, the recourse technique and the chance-constrained method are actually developed as methodologies for dealing with stochastic programming problems. Therefore, in this section, we will highlight stochastic programming problems to show the use of both methodologies for such problems, and we will see that the names of the recourse technique and the chance-constrained method will become clearer in this context.

In the context of stochastic programming, there are three fields of study [160]:

1. Distribution problems.
2. Stochastic programming with recourse.
3. Chance-constrained programming.

To show the differences between these fields of study, we will consider a linear programming problem:

$$\begin{aligned} \min_{x} \quad & z = c' x \qquad \text{(P1)} \\ \text{s.t.} \quad & A x \geq b \\ & x \geq 0, \end{aligned}$$

where A is an $m \times n$ matrix, c and x are n-vectors, and b is an m-vector. This linear programming problem becomes a stochastic linear programming problem if one or more of the coefficients from the set $\{A, b, c\}$ are random variables defined on the probability space $(\Omega, \mathcal{F}, \mu)$, where Ω is a Borel set, $\mathcal{F}$ is the σ-algebra of all the Borel subsets of Ω, and μ is a probability measure for $\mathcal{F}$.

Before we describe the second and third fields, we need to start the discussion from the first one since it can illustrate the essence of a stochastic programming problem in general. In distribution problems, the probability density function of the optimum $z^\star$ is sought. $z^\star$ will have different values depending upon the realized values of the set $\{A, b, c\}$. It means that $z^\star$ and the optimal decision vector $x^\star$ are random variables themselves. This type of problem was first formulated by Tintner [170] as a problem of "passive stochastic programming", or "wait and see" problem as defined later by Madansky [99]. In other words, the decision maker in this problem is passive and has to wait until the random coefficients are realized, and then solve the corresponding deterministic linear programming problem. However, before having the exact values of the coefficients, we sometimes need to know the distribution function, or certain typical measures of $z^\star$ and $x^\star$, or even the interval where the probability density function of $z^\star$ lies, subject to some *a priori* distribution functions of the set $\{A, b, c\}$.

A complete presentation of distribution problems for stochastic linear programming was given by Bereanu [14], together with numerical methods and computational experiences in solving such problems. It should be pointed out that the formulation of distribution problems simply allows the use of Monte Carlo methods. A simple procedure for obtaining the density function for $z^\star$ and $x^\star$ is given below:

1. Generate a realization of $\{A, b, c\}$ from their corresponding distribution functions.
2. Solve the deterministic linear programming problem (P1).
3. Collect and store the results: $z^\star$ and $x^\star$.

4. If the number of realizations used is less than the desired number, N, then go back to the second step, otherwise evaluate the next step.

5. Perform statistical analysis, such as plotting the empirical probability density functions of $z^\star$ and $x^\star$.

This simulation will generate $z^\star$ and $x^\star$ as random variables defined on an induced probability space $(\Omega^\star, \mathcal{F}^\star, \mu^\star)$ [160], where

$$\begin{aligned} \Omega^\star &= \{\omega | D(\omega) \neq \emptyset\} \subseteq \Omega, \\ \mathcal{F}^\star &= \mathcal{F} \cap \Omega^\star, \\ \mu^\star(\mathcal{A}) &= \frac{\mu(\mathcal{A})}{\mu(\Omega^\star)}, \quad \mathcal{A} \in \mathcal{F}^\star, \end{aligned}$$

and $D(\omega)$ is defined as follows:

$$D(\omega) = \{x(\omega) | A(\omega)x(\omega) \leq b(\omega), x(\omega) \geq 0\}, \quad \omega \in \Omega.$$

Therefore, if for a set of specific realizations of $\{A, b, c\}$ the second step of the above procedure cannot produce any solution, the statistical results should be normalized accordingly. As an example, if M realizations of $\{A, b, c\}$ do not produce any solution, the mean value of $z^\star$ becomes

$$E(z^\star) = \frac{1}{N-M} \sum_{i=1}^{N-M} z_i^\star,$$

where $z_i^\star$, $i = 1, \ldots, N-M$ are the optimal values that obtained from $N-M$ realizations of $\{A, b, c\}$. Several examples of applying the Monte Carlo method to derive the distribution function of the optimal value were given by Sarper [139]. The formulation of distribution problems suggests that such problems are not decision problems in the sense that they are concerned with determining probability distributions of functions of those uncertain coefficients [176].

Now, let us consider a problem in which a decision should first be taken before the values of the random variables are known and then, after the random events have happened and their values are known, a second decision is chosen in order to minimize the "penalty" that may appear due to any infeasibility. This might be interpreted as a decision that must be found "here and now" by an "active" decision maker, before the actual values of the components in the model are known; when they become known, then a "recourse" choice or decision must be found which will minimize the "penalty". This type of problem is called stochastic programming with recourse, because it uses the recourse technique to make optimal decisions. Clearly, in this problem we have a two-stage decision making process, therefore, sometimes it is also known as two-stage programming under uncertainty.

The stochastic linear programming problem (P1), consequently, can be formulated as a stochastic programming with recourse problem,

$$\begin{aligned} \min_x \quad & z = E\left(c'(\omega)\, x + \min_{q^+(\omega), q^-(\omega)} d'(\omega)\, q^+(\omega)\right) \qquad \text{(P2)} \\ \text{s.t.} \quad & A(\omega)\, x - b(\omega) + q^+(\omega) - q^-(\omega) = 0 \\ & x \geq 0, \end{aligned}$$

$$q^+(\omega) \geq 0,$$
$$q^-(\omega) \geq 0,$$

where ω is an element of the probability space. To simplify the notation of the problem, vectors $q^+(\omega)$ and $q^-(\omega)$ can be combined together into one vector $y(\omega)$. Furthermore, the formulation (P2) can be generalized as follows:

$$\begin{aligned} \min_x \quad & z = E\left(c'(\omega)\, x + \min_{y(\omega)} d'(\omega)\, y(\omega)\right) \\ \text{s.t.} \quad & A(\omega)\, x - b(\omega) + W(\omega)\, y(\omega) = 0 \\ & x \geq 0, \\ & y(\omega) \geq 0. \end{aligned} \qquad \text{(P3)}$$

Formulation (P2) is actually a special case of formulation (P3) when the matrix W is deterministic, and equals to $[I, -I]$. This special case of problem is called stochastic programming with simple recourse. If the matrix W is deterministic, the problem is known as problem with fixed recourse. Furthermore, if the matrix W is deterministic and such that the space of $\{Wy | y \geq 0\}$ is equal to the space of b, the problem is called problem with complete recourse. From these definitions, it is clear that a simple recourse problem is a special case of the class of complete recourse problems which in turn is a special class of fixed recourse problems.

A stochastic programming problem with recourse can be seen as an extension of the distribution problem, because for a specific value of the decision vector, x^0, we can solve the following distribution problem:

$$\begin{aligned} \min_{y(\omega)} \quad & s(\omega; x^0) = d'(\omega)\, y(\omega) \\ \text{s.t.} \quad & A(\omega)\, x^0 - b(\omega) + W(\omega)\, y(\omega) = 0 \\ & y(\omega) \geq 0, \end{aligned}$$

to calculate the value of z

$$z = E\left(c'(\omega)\, x^0 + s^\star(\omega; x^0)\right),$$

where $s^\star(\omega; x^0)$ denotes a random variable that represents the optimal value of the above distribution problem. The optimal objective function of the recourse problem, $z^\star$, is the minimum of z at $x^0 = x^\star$,

$$\begin{aligned} z^\star &= \min_x E\left(c'(\omega)\, x + s^\star(\omega; x)\right) \\ &= \min_x \bar{c}'\, x + \bar{s}^\star(x), \end{aligned} \qquad (6.7)$$

where the bar sign denotes the expected value. In this formulation, we can use Monte Carlo methods, such as the simulated annealing technique, to search for the optimal objective function $z^\star$. Consequently, for each chosen point of x we have to solve a distribution problem. A collection of other numerical methods and their implementations which are based on the approximation and discretization of uncertain variables and parameters can be seen in Ermoliev and Wets' book [45]. It should be pointed out that equation (6.7) represents

the deterministic equivalent of the two stage stochastic linear programming problem. As an example of the use of the two stage stochastic programming in the area of chemical engineering, Halemane and Grossmann [60] described a natural application of the two stage stochastic programming problem to process design.

A different way to formulate the decision problem is to consider that the constraints should not always hold, but that they need to be satisfied for a given proportion of cases or, to put it differently, they should hold with given probabilities. Thus, the stochastic linear programming problem can be reformulated as follows:

$$\begin{array}{ll} \min\limits_{x} & z = E(c'(\omega)\, x) \qquad \text{(P4)} \\ \text{s.t.} & P(A(\omega)\, x \;\geq\; b(\omega)) \;\geq\; \alpha \\ & x \;\geq\; 0, \end{array}$$

where $P(\cdot)$ represents the probability function. Charnes and Cooper [31] called the above problem (P4) which optimizes the expectation of objective function an E-model, while a V-model is one which minimizes the variance. Furthermore, they added to this what they called a P-model, where it is required to maximize the probability that the objective function does not exceed a given constant.

For those three models, we can transform the problem into one where no probability enters, a deterministic equivalent [177]. It means that if we have a closed form of the probability function of interest, then we can substitute the probabilistic constraint by the corresponding closed form function. Another way to obtain a deterministic equivalent is to use the quantile rules. To illustrate the use of such rules, let us consider a simple probabilistic constraint induced by an inequality constraint,

$$P(ax \geq b) \;\geq\; \alpha, \tag{6.8}$$

Two cases will be dealt with, and each case has only one uncertain parameter:

1. b has a known distribution function $P(b \leq k) = F_b(k)$.
2. a has a known distribution function $P(a \leq k) = F_a(k)$.

For the first case, we can find the smallest value B_α such that $F_b(B_\alpha) = \alpha$, which can also be written

$$B_\alpha \;=\; F_b^{-1}(\alpha).$$

Clearly, the probabilistic constraint (6.8) above is equivalent to the non-probabilistic constraint,

$$a\, x \;\geq\; B_\alpha.$$

The following equivalences are obtained by an analogous argument [177]:

$$\begin{array}{lll} P(ax \leq b) \;\leq\; \alpha & \text{is equivalent to} & a\, x \;\geq\; B_{1-\alpha} \\ P(ax \geq b) \;\leq\; \alpha & \text{is equivalent to} & a\, x \;\leq\; B_\alpha \\ P(ax \leq b) \;\geq\; \alpha & \text{is equivalent to} & a\, x \;\leq\; B_{1-\alpha}. \end{array}$$

For the second case, we can define $A_{1-\alpha}$ to be the largest value of a for which

$$\begin{aligned} F_a(A_{1-\alpha}) &= 1-\alpha, \text{ or} \\ A_{1-\alpha} &= F_a^{-1}(1-\alpha). \end{aligned}$$

If $x \geq 0$, then we have

$$P(ax \geq A_{1-\alpha}x) \equiv P(a \geq A_{1-\alpha}) = \alpha,$$

which means that

$$P(ax \geq b) \geq \alpha \text{ is equivalent to } A_{1-\alpha}\, x \geq b.$$

Similar to the first case, we can also generate the following equivalences for the second case:

$$\begin{aligned} P(ax \leq b) &\leq \alpha \text{ is equivalent to } A_\alpha\, x \geq b \\ P(ax \geq b) &\leq \alpha \text{ is equivalent to } A_{1-\alpha}\, x \leq b \\ P(ax \leq b) &\geq \alpha \text{ is equivalent to } A_\alpha\, x \leq b. \end{aligned}$$

For some forms of density function, it might be possible that there is no closed form function of A_α or B_α. This factor will clearly make the problem more difficult to solve, let alone the nonlinearity factor that induced by the probabilistic constraints.

However, since many robust algorithms have been developed for dealing with deterministic models, the need for transforming stochastic inequality or the stochastic optimization models into their corresponding deterministic equivalent models always appears, and many attempts have been made to explore this possibility [45, 160, 188]. It should also be noted that the equivalence notion in the model transformations should not literally be equivalent. It is sometimes sufficient to have and work with the approximation to such deterministic equivalent models. Therefore, from this point, we will refer the deterministic equivalent model as the non-probabilistic representation of a stochastic model, and it can be a precise equivalent, or an approximation of it.

Chapter 7

Deterministic Equivalent Modeling Method

In the previous chapter, the possibility of using of deterministic equivalent modeling approaches for solving stochastic models has been mentioned. Central to these approaches is the application of the polynomial chaos expansion and series approximations of probability density functions to representing stochastic variables and constraints. This chapter will explore the use of those approximations together with the variational approach to transform stochastic models. The main theme of this chapter is, therefore, the development of a new approach termed the deterministic equivalent modeling method or DEMM.

Since many problems in chemical and environmental engineering systems consist of different types of stochastic models, we will explain the procedure for transforming both stochastic equality and inequality models in the first two sections. Nevertheless, to deal with an optimization model, we clearly must be able to transform not only the stochastic equality and inequality constraints, but also the stochastic objective function. Therefore, several important aspects of the transformation of a stochastic objective function will be highlighted in the third section. The last section will show a further use of DEMM for black-box stochastic models. In this case, we may apply the collocation method to generate a set of response points in the same fashion as they are produced by the response surface methodology.

7.1 Equality model transformation

The central notion for the transformation of stochastic equality models is the analogy between a random variable and a deterministic function. A deterministic function $f(t)$ maps a point from one deterministic space, S_1, onto the other, S_2,

$$f(t): \ S_1 \ \rightarrow \ S_2$$

similarly, a random variable $r(\omega)$ maps an element of probabilistic space Ω onto the real line R,

$$r(\omega): \ \Omega \ \rightarrow \ R.$$

This analogy suggests that we can construct an approximation of a random variable by using several different random variables. Such an idea actually has been described in chapter 4

as polynomial chaos expansions. However, it should be pointed out that the formulation of some attributes of deterministic functions may have different forms than that for random variables. As an example, we define a deterministic function $f(t)$ to be orthogonal to function $g(t)$ as follows:

$$\int_T f(t)\, g(t)\, dt \;=\; 0,$$

on the other hand, two random variables $x(\omega)$ and $y(\omega)$ are orthogonal if the expectation of their product is zero (see chapter 3),

$$\int_{X,Y} f_{xy}(x,y)\, x\, y\, dx\, dy \;=\; 0,$$

where $f_{xy}(x,y)$ is the joint probability density function of $x(\omega)$ and $y(\omega)$.

This different concept of the orthogonalization process lead to different types of variational approaches for deterministic and stochastic models. In the following, we will see the differences between conducting the variational approach for simple deterministic and stochastic equality models. Let us assume that we want to construct a deterministic unknown function $f(t)$ from a given equality model,

$$a(t)\, f(t) \;=\; b(t),$$

where $a(t)$ and $b(t)$ are known functions. Clearly, the solution is $f(t) = \frac{b(t)}{a(t)}$, however, we would like to use methods of weighted residuals or variational approach to construct that unknown function. In other words, we want to approximate $f(t)$ by using a set of basis functions $\{g_i(t)\}$,

$$f(t) \;\approx\; \sum_{i=1}^{N} f_i\, g_i(t),$$

where $\{f_i\}$ is a set of N unknown coefficients. If we put this approximation back into the equality model above, we will have an error or a residual function which is defined by

$$R_N(\mathbf{f},t) \;=\; a(t) \sum_{i=1}^{N} f_i\, g_i(t) \;-\; b(t),$$

where $\mathbf{f}$ is the vector of unknown coefficients. In Galerkin's approach, we can minimize the residual function by making it orthogonal to the approximation or basis functions space,

$$\int_T R_N(\mathbf{f},t)\, g_i(t)\, dt \;=\; 0;\;\; i = 1,\ldots,N.$$

In general, the methods of weighted residuals make the average of weighted residual function equal to zero [183],

$$\int_T R_N(\mathbf{f},t)\, W_i(t)\, dt \;=\; 0;\;\; i = 1,\ldots,N,$$

where $\{W_i(t)\}$ is a sequence of N weight functions, and for the case of Galerkin's approach it is the set of basis functions.

In this deterministic model, a set of N equations is formulated to calculate the value of $\mathbf{f}$. It also means that we do not have to deal with the solution function directly which should be defined at every value of t in T. We only need to deal with N unknown coefficients. Such a reduction in the dimensionality of the problem makes the variational approach attractive for solving both function and functional models.

Similarly, the use of the variational approach to stochastic equality models is clearly needed because with this approach we may not have to deal with random variables directly. Only deterministic coefficients in the representation must be considered. To illustrate such an idea, a simple stochastic equality model is considered,

$$a(\omega)\ x(\omega) \quad = \quad b(\omega), \tag{7.1}$$

where $a(\omega)$ and $b(\omega)$ are known random variables, and the goal is to compute random variable $x(\omega)$. To use the methods of weighted residuals, we need an approximation of $x(\omega)$ in terms of other known random variables. At this point, the polynomial chaos expansion comes into play (this expansion is explained in more detail in chapter 4). Therefore, a polynomial chaos expansion of random variable $x(\omega)$ can be constructed, and it consists of a set of polynomials of standard independent Gaussian random variables $\{\xi_i(\omega)\}$,

$$x(\omega) \quad \approx \quad \sum_{j=1}^{N} x_i\ H_j(\{\xi_i(\omega)\}),$$

where $H_i(\cdot)$ denotes the multi-dimensional Hermite polynomial of order i. To simplify the notation, let $\mathbf{x}$ represents the vector of unknown coefficients in the polynomial chaos expansion above. The residual random variable for this model can be written as follows:

$$R_N(\mathbf{x},\omega) \quad = \quad a(\omega) \sum_{j=1}^{N} x_j\ H_j(\{\xi_i(\omega)\}) - b(\omega).$$

Now, the orthogonalization process in the Galerkin's approach for this residual random variable should be the probabilistic orthogonalization,

$$\int_{\{\xi_i\}} f_{\{\xi_i\}}(\{\xi_i\})\ R_N(\mathbf{x},\omega)\ H_j(\{\xi_i(\omega)\})\ d\{\xi_i(\omega)\} \quad = \quad 0;\ \ j = 1,\ldots,N,$$

where $f_{\{\xi_i\}}(\{\xi_i\})$ is the joint probability density function of $\{\xi_i\}$. Clearly, for other methods of weighted residuals, the averaging process of weighted residuals should be taken as the expected value operation,

$$E\left(R_N(\mathbf{x},\omega)\ W_j(\omega)\right) \quad = \quad 0;\ \ j = 1,\ldots,N,$$

where $W_j(\omega)$ is the j-th weight random variable.

From the process of probabilistic orthogonalization or expected value operation above, we will have N deterministic equations in which the N unknowns are the deterministic coefficients in the polynomial chaos expansion of $x(\omega)$. By computing these N unknown coefficients, we will have an approximation to the random variable $x(\omega)$. In other words, one may say or accept that solving these N deterministic equations are somewhat "equivalent" to solving the stochastic equation (7.1). Therefore, the use of a combination of direct representations, such as polynomial chaos expansions, of a random variable, and the varia-

tional approach to generate a deterministic equivalent model of a stochastic equality model becomes a part of the proposed deterministic equivalent modeling method or DEMM.

There are two methods of weighted residuals which will be used in the DEMM: the Galerkin's and the collocation approaches. It has been mentioned previously that the Galerkin's approach applies the probabilistic orthogonalization to generate the deterministic equivalent models. To illustrate such a process, let us consider again the stochastic equality model defined by equation (7.1), and now the coefficients $a(\omega)$ and $b(\omega)$ are random variables with polynomial chaos expansions given by

$$\begin{aligned} a &= a_1 + a_2\xi_1 + a_3\xi_2 + a_4(\xi_1^2 - 1) + a_5\xi_1\xi_2 + a_6(\xi_2^2 - 1) + \cdots \\ b &= b_1 + b_2\xi_1 + b_3\xi_2 + b_4(\xi_1^2 - 1) + b_5\xi_1\xi_2 + b_6(\xi_2^2 - 1) + \cdots, \end{aligned}$$

where ξ_1 and ξ_2 are independent standard Gaussian random variables. Without loss of generality, we may assume that $a(\omega)$ and $b(\omega)$ are independent Gaussian random variables, then their polynomial chaos expansions become

$$\begin{aligned} a &= a_1 + a_2\xi_1 \\ b &= b_1 + b_3\xi_2. \end{aligned}$$

Similarly, we must expand the random variable $x(\omega)$ into its polynomial chaos expansion,

$$x = x_1 + x_2\xi_1 + x_3\xi_2 + x_4(\xi_1^2 - 1) + x_5\xi_1\xi_2 + x_6(\xi_2^2 - 1) + \cdots,$$

and for this example, let us assume that six terms are sufficient for $x(\omega)$ (the polynomial chaos expansion is converged in the mean-square sense [24]). The residual random variable can now be written as:

$$\begin{aligned} R_6(\mathbf{x},\omega) = {} & (a_1 + a_2\xi_1)\,(x_1 + x_2\xi_1 + x_3\xi_2 + x_4(\xi_1^2 - 1) + x_5\xi_1\xi_2 + x_6(\xi_2^2 - 1)) - \\ & (b_1 + b_3\xi_2), \end{aligned}$$

and the orthogonalization process becomes

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{\xi_1}(\xi_1)\, f_{\xi_2}(\xi_2)\, H_j(\xi_1,\xi_2)\, R_6(\mathbf{x},\omega)\, d\xi_1\, d\xi_2 = 0; \quad j = 1,\ldots,6, \qquad (7.2)$$

where $f_{\xi_1}(\xi_1)$ and $f_{\xi_2}(\xi_2)$ are the Gaussian density functions of ξ_1 and ξ_2 respectively, and $H_j(\xi_1,\xi_2)$ denotes the j-th order of Hermite polynomial of ξ_1 and ξ_2 used in the polynomial chaos expansion,

$$\begin{aligned} H_1(\xi_1,\xi_2) &= 1, \\ H_2(\xi_1,\xi_2) &= \xi_1, \\ H_3(\xi_1,\xi_2) &= \xi_2, \\ H_4(\xi_1,\xi_2) &= \xi_1^2 - 1, \\ H_5(\xi_1,\xi_2) &= \xi_1\xi_2, \\ H_6(\xi_1,\xi_2) &= \xi_2^2 - 1. \end{aligned}$$

We can also interpret the equation (7.2) as the expected value of a polynomial function of

two independent Gaussian random variables ξ_1 and ξ_2:

$$E(g_j(\xi_1, \xi_2; \mathbf{x})) = 0; \quad j = 1, \ldots, 6,$$

where $g_j(\xi_1, \xi_2; \mathbf{x}) = H_j(\xi_1, \xi_2)\, R_6(\mathbf{x}, \omega)$, and it contains only polynomials of ξ_1 and ξ_2.

Since the expected value of a polynomial function of Gaussian random variables is readily calculated using the formula (4.7),

$$E(\xi^n) = \begin{cases} 0 & n = 2k+1 \\ 1 \cdot 3 \cdots (n-1) & n = 2k, \end{cases}$$

we do not have to symbolically compute multi-dimensional integrations in equation (7.2), but only need to make symbolic substitutions for all polynomials of ξ_1 and ξ_2 according to the formula above. As an example, for the case of $j = 2$, we want to evaluate

$$\begin{aligned}
E\left(\xi_1 R_6(\mathbf{x}, \omega)\right) &= 0 \\
E\left(\xi_1 a_1 (x_1 + x_2\xi_1 + x_3\xi_2 + x_4(\xi_1^2 - 1) + x_5\xi_1\xi_2 + x_6(\xi_2^2 - 1))\right) + & \\
E\left(\xi_1^2 a_2 (x_1 + x_2\xi_1 + x_3\xi_2 + x_4(\xi_1^2 - 1) + x_5\xi_1\xi_2 + x_6(\xi_2^2 - 1))\right) - & \\
E\left(\xi_1 b_1 + \xi_1\xi_2 b_3\right) &= 0 \\
a_1 x_1 E(\xi_1) + a_1 x_2 E(\xi_1^2) + a_1 x_3 E(\xi_1)E(\xi_2) + a_1 x_4 \left(E(\xi_1^3) - E(\xi_1)\right) + & \\
a_1 x_5 E(\xi_1^2)E(\xi_2) + a_1 x_6 \left(E(\xi_1)E(\xi_2^2) - E(\xi_1)\right) + a_2 x_1 E(\xi_1^2) + a_2 x_2 E(\xi_1^3) + & \\
a_2 x_3 E(\xi_1^2)E(\xi_2) + a_2 x_4 \left(E(\xi_1^4) - E(\xi_1^2)\right) + a_2 x_5 E(\xi_1^3)E(\xi_2) + & \\
a_2 x_6 \left(E(\xi_1^2)E(\xi_2^2) - E(\xi_1^2)\right) - b_1 E(\xi_1) - b_3 E(\xi_1)E(\xi_2) &= 0,
\end{aligned}$$

and after the process of symbolic substitution or manipulation, it finally becomes

$$a_1 x_2 + a_2 x_1 + 2a_2 x_4 = 0.$$

The resulting deterministic equivalent equations for $j = 1, \ldots, 6$ are in the following form,

$$\begin{bmatrix} a_1 & a_2 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & 0 & 2a_2 & 0 & 0 \\ 0 & 0 & a_1 & 0 & a_2 & 0 \\ 0 & 2a_2 & 0 & 2a_1 & 0 & 0 \\ 0 & 0 & a_2 & 0 & a_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} b_1 \\ 0 \\ b_3 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.3}$$

and the solution for this set of equations are

$$\begin{aligned}
x_1 &= \frac{b_1}{a_1} \times \frac{2a_2^2 - a_1^2}{3a_2^2 - a_1^2} \\
x_2 &= \frac{b_1 a_2}{3a_2^2 - a_1^2} \\
x_3 &= \frac{b_3 a_1}{a_1^2 - a_2^2}
\end{aligned}$$

$$
\begin{aligned}
x_4 &= \frac{b_1 a_2^2}{a_1(a_1^2 - 3a_2^2)} \\
x_5 &= \frac{b_3 a_2}{a_2^2 - a_1^2} \\
x_6 &= 0.
\end{aligned}
$$

This result suggests several things:

- $x(\omega)$ is not a quadratic function of $\xi_2(\omega)$ because the value of x_6 equals to zero;
- the mean value of $x(\omega)$, which is represented by its first coefficient x_1, does not depend on the value of the standard deviation of random variable $b(\omega)$, b_3;
- when a_2 equals to zero, the result becomes exact, $x(\omega) = \frac{b_1}{a_1} + \frac{b_3}{a_1}\xi_2(\omega)$;
- when both a_2 and b_3 equal to zero, the result becomes the correct solution of the deterministic problem, $x = \frac{b_1}{a_1}$.
- when both a_1 and b_1 equal to zero, the moments of $x(\omega)$, such as its mean value, x_1, and its variance, $x_2^2 + x_3^2 + 2x_4^2 + x_5^2$, are undefined because the first and the fourth coefficients are undefined;
- This six-term approximation only works when the three conditions, $a_2^2 \neq a_1^2$, $2a_2^2 \neq a_1^2$, and $3a_2^2 \neq a_1^2$, are satisfied.

The first two points are self-explanatory from observing the structure of the model. The third and fourth points show that the deterministic equivalent model given by equation (7.3) can capture the essence of the model. A ratio of two Gaussian random variables with zero mean values is analytically known to have a Cauchy distribution, which also means that the ratio has undefined moments values. Therefore, the fifth point above indicates that the result of DEMM reaches the same conclusion: the distribution of $x(\omega)$ has undefined moments values for such a case. The last point illustrates the limitation of the approximation scheme induced by DEMM. The number and the form of those limiting conditions may change for different number of terms used for the approximation.

In general, DEMM with Galerkin's approach can be used in the equality model development phase to provide the users of the model an option to conduct uncertainty analyses of the model whenever it is necessary. In other words, with the corresponding deterministic equivalent model at hand, the user of the model can choose to treat the model as either a deterministic or a stochastic model. For a given equality model that has polynomial forms of non-linearity for uncertain parameters and variables in the model, the flow diagram in figure 7-1 presents the procedure to develop the deterministic equivalence of that model.

Other non-linearity cases can be handled in two different ways. In the first approach, all non-linear-non-polynomial forms involving uncertain parameters and variables in the stochastic equality model should be transformed into or approximated by polynomial functions. Techniques like Adomian polynomials [2], Padé approximants [128], or the transformation technique for ordinary differential equations [79] can be applied to generate such polynomial approximations. Different approaches are to use numerical quadrature methods in the variational process, such as the Gauss quadrature method [200], or to use the collocation method instead of the Galerkin's method. This class of approaches treats the stochastic model as a set of deterministic problems because it solves the deterministic problem at several chosen values of uncertain parameters. Clearly, such a scheme is also useful

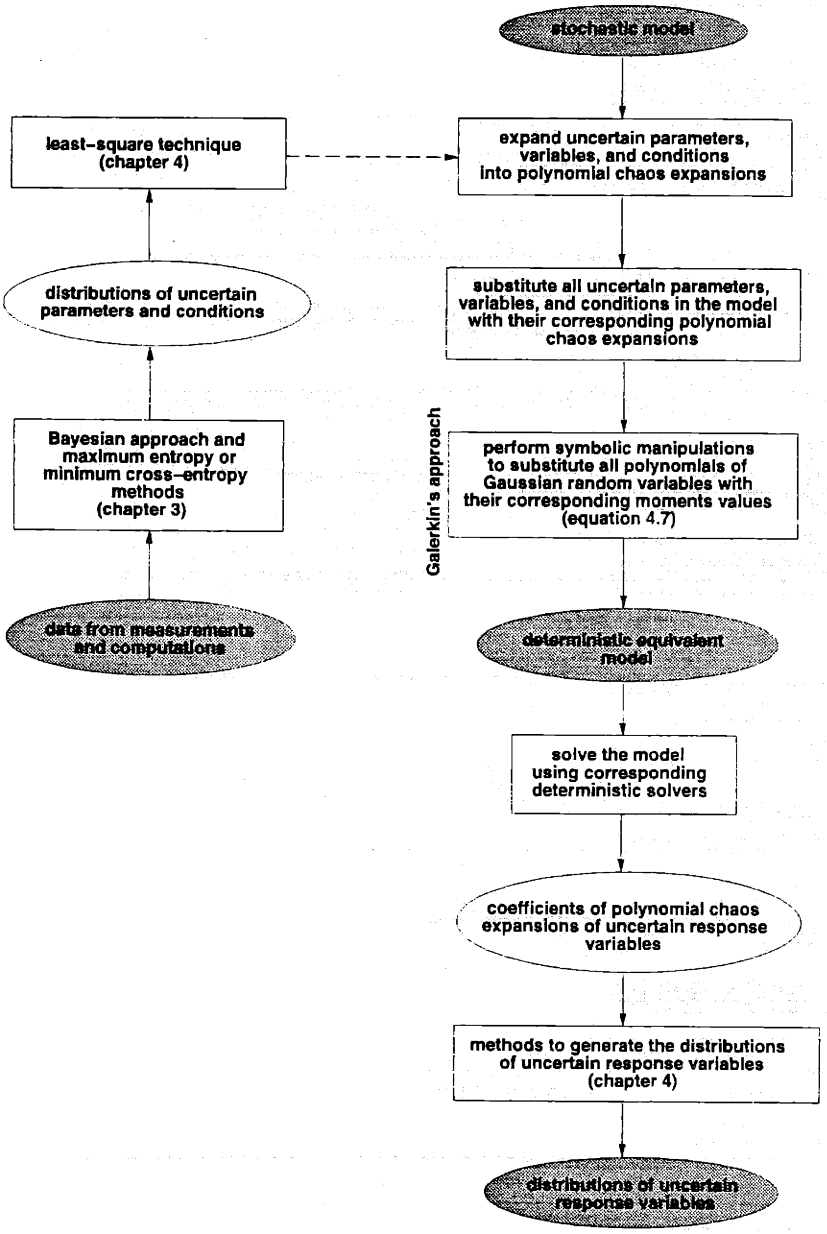


Figure 7-1: Flow diagram of the DEMM for stochastic equality models.

in dealing with implicit or black-box type stochastic models. Therefore, the use of the collocation method to those models will be discussed in the last section of this chapter.

7.2 Inequality model transformation

In chapter 6, we have seen two common methods for dealing with stochastic inequality models. The extent of violation is the primary objective for the recourse technique, while the chance-constrained method measures the probability or frequency of violation. Since these two main methods differ in their formulation of stochastic inequalities, the DEMM has to deal with each formulation differently. As an example, let us consider a simple problem,

$$x(\omega) \leq 0, \tag{7.4}$$

where $x(\omega)$ is a random variable. The recourse formulation decomposes the slack variable of that stochastic constraint into a pair of positive-valued slack variables,

$$\begin{aligned} x(\omega) + q^+(\omega) - q^-(\omega) &= 0, \\ q^+(\omega)\, q^-(\omega) &= 0, \end{aligned}$$

which can also be written as follows

$$\begin{aligned} x(\omega) + s_1^2(\omega) - s_2^2(\omega) &= 0, \\ s_1^2(\omega)\, s_2^2(\omega) &= 0, \end{aligned} \tag{7.5}$$

where $q^+(\omega) = s_1^2(\omega)$ and $q^-(\omega) = s_2^2(\omega)$. If we can represent $x(\omega)$, $s_1(\omega)$ and $s_2(\omega)$ by polynomial chaos expansions, we can clearly transform formulation (7.5) into its deterministic equivalence since we now have stochastic equations. The case of $P(x(\omega) \leq 0) \approx 1$ is a special case of formulation (7.5) which can be obtained by setting $s_2^2(\omega) = 0$. Therefore, in general we can transform an arbitrary stochastic inequality model into its deterministic equivalence by the use of the recourse formulation above and DEMM. However, it should be pointed out that such a formulation changes the structure of the model: the inequality model is converted into an equality model. A different type of formulation which introduces an equality model as well as conserves the structure of the inequality model is available through the use of chance-constrained method. For the problem under consideration, we may have the following formulation,

$$x(\omega) - s(\omega) = 0 \tag{7.6}$$

$$P(s(\omega) \leq \alpha) \leq 1 - \beta, \tag{7.7}$$

where $s(\omega)$ denotes the slack random variable, and $1 - \beta$ represents the least probability that $x(\omega)$ is less than α. Since the cumulative distribution function, $P(\cdot)$, is actually a deterministic function that depends only on the value of α, equation (7.7) may be referred to the deterministic equivalence of the stochastic inequality model (7.4). For this case, the value of α is known to be 0, but the value of β remains undefined. Depending on the structure of the problem, β can be a known parameter or an unknown variable of interest. If we know the distribution function of $x(\omega)$, the value of $P(s(\omega) \leq \alpha)$ can be computed. On the other hand, if the distribution function of $x(\omega)$ is not available, the value of β must be chosen to determine the shape or the values of parameters of that distribution.

Clearly, DEMM can transform the induced stochastic equality model (7.6) into its deterministic equivalence. Furthermore, the coefficients of the polynomial chaos expansion of $s(\omega)$ may be used to make an approximation to the required cumulative distribution function $P(s(\omega) \leq \alpha)$. The key goal now is to apply the indirect representation of the

slack random variable $s(\omega)$ to obtain an approximation to that cumulative distribution function. Chapter 4 has discussed some indirect representations of random variables. Since the application of the orthogonal expansion approach and the linear combination of distributions method is relatively straightforward and quite general, both methods are selected to indirectly represent the slack random variables in the stochastic inequality model.

In the following, we will see the use of the Edgeworth series expansion as an instance of the orthogonal expansion approach to approximate the joint cumulative distribution function of a set of slack random variables. The Edgeworth series expansion is a modified version of the Gram-Charlier series expansion [107]. The convergence of this series expansion depends on the difference of the shape of the limiting distribution used in the expansion and the shape of the approximated distribution. The most common limiting distribution for the Edgeworth series expansion is a joint Gaussian distribution. For a k-vector of slack variables, $\mathbf{u}$, with covariance matrix $\boldsymbol{\Sigma}$, its probability density function $f_{\mathbf{u}}(\mathbf{u})$ can be approximated by the following Edgeworth series expansion,

$$\hat{f}_{\mathbf{u}}(\mathbf{u}) \;=\; \exp(-\frac{1}{2}\,\mathbf{u}'\,\boldsymbol{\Sigma}^{-1}\,\mathbf{u}) \sum_{\mathbf{m}=\mathbf{0}}^{\infty} a_{\mathbf{m}}\, H(\mathbf{u},\mathbf{m},\boldsymbol{\Sigma}^{-1}),$$

where $\exp(-\frac{1}{2}\,\mathbf{u}'\,\boldsymbol{\Sigma}^{-1}\,\mathbf{u})$ is the limiting distribution, $H(\mathbf{u},\mathbf{m},\boldsymbol{\Sigma}^{-1})$ represents the multi-dimensional Hermite polynomial, and the coefficients $a_{\mathbf{m}}$ are calculated from the values of joint moments of the slack random variables $\mathbf{u}$. As an example, the value of coefficient $a_{\mathbf{k}}$ is obtained from taking the expected value of $H(\mathbf{u},\mathbf{k},\boldsymbol{\Sigma}^{-1})$ which in turn is a function of joint moments of $\mathbf{u}$,

$$a_{\mathbf{k}} \;=\; \int_{-\infty}^{\infty} f_{\mathbf{u}}(\mathbf{u})\, H(\mathbf{u},\mathbf{k},\boldsymbol{\Sigma}^{-1})\, d\mathbf{u}. \tag{7.8}$$

The probability of $\mathbf{u} \leq \mathbf{c}$ becomes

$$P(\mathbf{u} \leq \mathbf{c}) \;=\; \int_{-\infty}^{\mathbf{c}} \exp(-\frac{1}{2}\,\mathbf{x}'\,\boldsymbol{\Sigma}^{-1}\,\mathbf{x}) \sum_{\mathbf{m}=\mathbf{0}}^{\infty} a_{\mathbf{m}}\, H(\mathbf{x},\mathbf{m},\boldsymbol{\Sigma}^{-1})\, d\mathbf{x}. \tag{7.9}$$

To clarify the notation for multi-dimensional Hermite polynomials, let us consider a k-dimensional space, and in order to write multivariate series concisely, let

$$\boldsymbol{\theta} * *\mathbf{m} \;=\; \prod_{j=1}^{k} \theta_j^{m_j}$$

and

$$m_j^{\circ} \;=\; m_j - \sum_{i=1}^{k} e_{i,j}; \quad \forall\, j \,\in\, [1,\ldots,k]$$

for any k-vector $\boldsymbol{\theta}$ and k-vector of integers $\mathbf{m}$, and $\mathbf{e}_i$ denotes the i-th unit vector in k-space. The Hermite polynomials $H(\mathbf{y},\mathbf{m},\mathbf{A})$ then can be defined as follows:

$$H(\mathbf{y},\mathbf{m},\mathbf{A}) \;=\; \exp(\frac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y}) \left(-\frac{\partial}{\partial \mathbf{y}}\right) * *\mathbf{m} \left\{\exp(-\frac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\right\}$$

The following results of observations 1 and 2 can be used to help transform the integration (7.9) into a series of summations for the case of $m_j \geq 1; \forall\, j \in [1, \ldots, k]$.

Observation 1

$$\int_0^x e^{-y^2}\, H_n(y)\, dy \;=\; H_{n-1}(0) \;-\; e^{-x^2}\, H_{n-1}(x)$$

From the definition of Hermite polynomials, $H_n(y) = (-1)^n\, e^{y^2}\, \frac{d^n}{dy^n}\left(e^{-y^2}\right)$, *the above observation becomes:*

$$\begin{aligned}
\int_0^x e^{-y^2}\,(-1)^n\, e^{y^2}\, \frac{d^n}{dy^n}\left(e^{-y^2}\right) dy &= (-1)^n\, \frac{d^{n-1}}{dy^{n-1}}\left(e^{-y^2}\right)\Big|_0^x \\
&= -\,e^{-y^2}\,(-1)^{n-1}\, e^{y^2}\, \frac{d^{n-1}}{dy^{n-1}}\left(e^{-y^2}\right)\Big|_0^x \\
&= -\,e^{-y^2}\, H_{n-1}(y)\,\Big|_0^x \\
&= H_{n-1}(0) \;-\; e^{-x^2}\, H_{n-1}(x)
\end{aligned}$$

□

The above observation can be redefined for the k-space case.

Observation 2

$$\begin{aligned}
\int_0^{\mathbf{x}} \exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\, H(\mathbf{y}, \mathbf{m}, \mathbf{A})\, d\mathbf{y} &= (-1)^k\, \exp(-\tfrac{1}{2}\,\mathbf{x}'\,\mathbf{A}\,\mathbf{x})\, H(\mathbf{x}, \mathbf{m}^\circ, \mathbf{A}) \;+ \\
&\quad (-1)^{k+1}\, H(\mathbf{0}, \mathbf{m}^\circ, \mathbf{A}).
\end{aligned}$$

This observation is derived from applying the definition of Hermite polynomials in k*-space.*

$$\int_0^{\mathbf{x}} \exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\, \exp(\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\, \left(-\frac{\partial}{\partial \mathbf{y}}\right) **\mathbf{m}\, \left\{\exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\right\}\, d\mathbf{y} =$$

$$\int_0^{\mathbf{x}} \left(-\frac{\partial}{\partial \mathbf{y}}\right) **\mathbf{m}\, \left\{\exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\right\}\, d\mathbf{y} =$$

$$(-1)^k\, \left(-\frac{\partial}{\partial \mathbf{y}}\right) **\mathbf{m}^\circ\, \left\{\exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\right\}\Big|_0^{\mathbf{x}} =$$

$$(-1)^k\, \exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\, \exp(\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\, \left(-\frac{\partial}{\partial \mathbf{y}}\right) **\mathbf{m}^\circ\, \left\{\exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\right\}\Big|_0^{\mathbf{x}} =$$

$$(-1)^k\, \exp(-\tfrac{1}{2}\,\mathbf{y}'\,\mathbf{A}\,\mathbf{y})\, H(\mathbf{y}, \mathbf{m}^\circ, \mathbf{A})\Big|_0^{\mathbf{x}} =$$

$$(-1)^k\, \exp(-\tfrac{1}{2}\,\mathbf{x}'\,\mathbf{A}\,\mathbf{x})\, H(\mathbf{x}, \mathbf{m}^\circ, \mathbf{A}) \;+\; (-1)^{k+1}\, H(\mathbf{0}, \mathbf{m}^\circ, \mathbf{A})$$

□

A recursive equation to calculate Hermite polynomials in k- space is available as fol-

lows.

$$H(\mathbf{y},\mathbf{m},\mathbf{A}) \;=\; -\frac{\partial}{\partial y_i} H(\mathbf{y},\mathbf{m}^*,\mathbf{A}) \;+\; \mathbf{y}'\,\mathbf{A}\,\mathbf{e}_i\; H(\mathbf{y},\mathbf{m}^*,\mathbf{A})$$

where

$$m_j^* \;=\; m_j \;-\; e_{i,j};\quad \forall\, j \;\in\; [1,\ldots,k].$$

If a similar orthogonal expansion is applied to the limiting distribution, the computation of multi-dimensional integrations in the case $m_j = 0$ may be unnecessary. The following equation defines an orthogonal expansion approximation to the limiting distribution $f_{\mathbf{u}}^l(\mathbf{u})$,

$$f_{\mathbf{u}}^l(\mathbf{u}) \;=\; \left(\prod_{i=1}^{N} p(u_i)\right) \sum_{j=0}^{\infty} a_j\, Q_j(\mathbf{u})$$

where $Q_j(\mathbf{u})$ is the j-th multi-dimensional orthonormal polynomial, such that

$$\int_{\mathbf{D}} \left(\prod_{i=1}^{N} p(u_i)\right) Q_m(\mathbf{u})\, Q_n(\mathbf{u})\, d\mathbf{u} \;=\; \delta_{mn}.$$

Since the limiting distribution is Gaussian, $p(u_i)$ becomes a normalized univariate Gaussian distribution, and $Q_j(\mathbf{u})$ is the j-th standard multi-variate Hermite polynomial. Therefore, the limiting distribution can be written down as follows:

$$\begin{aligned}\exp(-\frac{1}{2}\,\mathbf{u}'\,\boldsymbol{\Sigma}^{-1}\,\mathbf{u}) \;=\; & \left(\prod_{i_1=1}^{N} e^{\frac{-u_{i_1}^2}{2}}\right)[a_0 + \sum_{i_1=1}^{N}\sum_{i_2=1}^{\infty} a_{i_1,i_2} H_{i_2}(u_{i_1}) + \\ & \sum_{i_1=1}^{N-1}\sum_{i_2=i_1+1}^{N}\sum_{i_3=1}^{\infty}\sum_{i_4=1}^{\infty} a_{i_1,i_2,i_3,i_4} H_{i_3}(u_{i_1}) H_{i_4}(u_{i_2}) + \cdots] \quad (7.10)\end{aligned}$$

In this case, the result from observation 1 can be used to transform the integration of limiting distribution into series of summations,

$$\begin{aligned}P^l(\mathbf{u}\le\mathbf{c}) \;=\; & \left(\prod_{i_1=1}^{N}(0.5+0.5Erf(0.707107c_{i_1})\right) a_0 \\ & + \left(\frac{-1}{\sqrt{2\pi}}\right)\sum_{i_1=1}^{N}\sum_{i_2=1}^{\infty}\Big(\prod_{i_3\in N\setminus\{i_1\}}(0.5+0.5 \\ & Erf(0.707107c_{i_3}))\Big)\; e^{\frac{-c_{i_1}^2}{2}} a_{i_1 i_2} H_{i_2-1}(c_{i_1}) + \left(\frac{-1}{\sqrt{2\pi}}\right)^2 \\ & \sum_{i_1=1}^{N-1}\sum_{i_2=i_1+1}^{N}\sum_{i_3=1}^{\infty}\sum_{i_4=1}^{\infty}\Big(\prod_{i_5\in N\setminus\{i_1,i_2\}}(0.5+0.5 \\ & Erf(0.707107c_{i_5}))\Big)\; e^{\frac{-(c_{i_1}^2+c_{i_2}^2)}{2}} a_{i_1 i_2 i_3 i_4} H_{i_3-1}(c_{i_1}) \\ & H_{i_4-1}(c_{i_2}) + \cdots,\end{aligned}$$

where $P^l(\cdot)$ is the probability of the limiting distribution, and c_i is the value of the i-th element of vector $\mathbf{c}$. Since it is quite often that the first term, $\mathbf{m} = \mathbf{0}$, of the expansion produces a good approximation of the joint probability of slack variables, the use of equation (7.10) for many cases is sufficient.

The linear combination of distributions method generates an optimal estimator of distribution from its moments values. To illustrate this concept (see also chapter 4), let us first consider a single slack random variable u with mean value m, variance σ^2, and central moments m_n and dimensionless central moments $\gamma_n = \frac{m_n}{\sigma^n}$ are finite and available for orders n smaller than N. The cumulative distribution function of this slack random variable, F, can be estimated by an optimal linear combination of prescribed cumulative distribution functions [58],

$$\hat{F} = \sum_{i=1}^{L} k_i \hat{F}_i,$$

where $\hat{F}_i$ represents the prescribed distribution with the same mean and variance as F, and k_i denotes the weighting factor that satisfies the following conditions

$$\begin{aligned} k_i &\geq 0; \quad i = 1, \ldots, L \\ \sum_{i=1}^{L} k_i &= 1. \end{aligned}$$

Since all $\hat{F}_i$ have the same mean and variance as F, the distribution $\hat{F}$ is equal to F in the second moment sense for any admissible values of weighting factors k_i. The optimal weighting factors k_i^* minimize the objective function

$$\epsilon_N = \sum_{n=3}^{N} (\gamma_n - \hat{\gamma}_n)^2,$$

and $\hat{\gamma}_n$ in that objective function is calculated from a linear combination of dimensionless central moments $\hat{\gamma}_{n,i}$ of prescribed distributions $\hat{F}_i$,

$$\hat{\gamma}_n = \sum_{i=1}^{L} k_i \hat{\gamma}_{n,i}.$$

The optimal values of weighting factors are obtained by solving Lagrange equations of the unconstrained optimization problem (see chapter 4).

A similar approach can be applied to the case of a vector of slack random variables $\mathbf{u}$ with mean vector $\mathbf{m}$, covariance matrix $\mathbf{\Sigma}$. Its cumulative distribution function, $\mathbf{F}$, may be estimated by using prescribed cumulative distribution functions

$$\hat{\mathbf{F}} = \sum_{i=1}^{L} k_i \hat{\mathbf{F}}_i,$$

where $\hat{\mathbf{F}}_i$ has the same mean vector and covariance matrix as $\mathbf{F}$. The optimal weighting

factors for this case are obtained by minimizing the following objective function

$$\epsilon_{\mathbf{N}} = \sum_{\mathbf{n=3}}^{\mathbf{N}} (\gamma_{\mathbf{n}} - \hat{\gamma}_{\mathbf{n}})^2 ,$$

where $\gamma_{\mathbf{n}}$ is the $\mathbf{n}$-th dimensionless central moment defined by

$$\gamma_{\mathbf{n}} = E\left(\prod_{j=1}^{J} \frac{(u_j - m_j)^{n_j}}{\sigma_{u_j}^{n_j}} \right),$$

where u_j, m_j, and n_j are the j-th element of vector $\mathbf{u}$, $\mathbf{m}$, and $\mathbf{n}$ respectively, J is the dimension of vector $\mathbf{u}$, and σ_{u_j} denotes the standard deviation of slack random variable u_j. Therefore, an optimal linear combination of prescribed cumulative distribution functions for a vector of slack random variables $\mathbf{u}$ can provide an estimation of the probability that $\mathbf{u} \leq \mathbf{c}$

$$P(\mathbf{u} \leq \mathbf{c}) = \sum_{i=1}^{L} k_i^* \, \hat{\mathbf{F}}_i(\mathbf{c}),$$

where $\hat{\mathbf{F}}_i(\mathbf{c})$ denotes the value of the i-th prescribed cumulative distribution function at point $\mathbf{c}$.

The prescribed distribution $\hat{\mathbf{F}}_i$ with mean vector $\mathbf{m}$ and variance matrix $\mathbf{\Sigma}$ may further be approximated by orthogonal expansions such as the Edgeworth series expansion. A specific orthogonal expansion can be derived for a prescribed probability density function $\hat{\mathbf{f}}_i$ of random vector $\mathbf{x}$ with mean vector $\mathbf{m}$ and variance matrix $\mathbf{\Sigma}$,

$$\hat{\mathbf{f}}_i = \left(\prod_{j=1}^{J} f_j \right) \left(1 + \sum_{k=1}^{\infty} \mathbf{q}_k^T \, \mathbf{R}^k \, \mathbf{q}_k \right), \tag{7.11}$$

where f_j is the probability density function of the j-th element of $\mathbf{x}$ with mean value m_j and variance $\Sigma_{j,j}$, and $\mathbf{q}_k$ denotes the vector of orthonormal polynomials of order k in which each element satisfies the condition:

$$\int_{x_j} f_j \; q_r(x_j) \; q_s(x_j) \; dx_j = \delta_{rs},$$

where $q_r(x_j)$ is the orthonormal polynomial of order r with weighting factor f_j. $\mathbf{R}^k$ is an upper-triangular matrix where its element is calculated from the corresponding values of elements in the covariance matrix $\mathbf{\Sigma}$,

$$R_{i,j}^k = \begin{cases} \frac{\Sigma_{i,j}^k}{\Sigma_{i,i}^k}; & j > i \\ 0; & j \leq i. \end{cases}$$

As an example, let us consider a two-dimensional case, $J = 2$. The probability density function of $\mathbf{x}$ can be written as follows:

$$\hat{\mathbf{f}}_i = f_1 f_2 \left(1 + \sum_{k=1}^{\infty} R^k q_k(x_1) q_k(x_2) \right),$$

where $R^k = \frac{\Sigma^k_{1,2}}{\Sigma^k_{1,1}}$. This representation actually satisfies the following six conditions [12]:

1. $\hat{\mathbf{f}}_i \geq 0$.
2. $\int_{x_1} \int_{x_2} \hat{\mathbf{f}}_i \, dx_2 \, dx_1 = 1$.
3. $\int_{x_1} \hat{\mathbf{f}}_i \, dx_1 = f_2$ and $\int_{x_2} \hat{\mathbf{f}}_i \, dx_2 = f_1$.
4. $\frac{\int_{x_1} \int_{x_2} (x_1 x_2 - E(x_1)E(x_2)) \hat{\mathbf{f}}_i \, dx_2 \, dx_1}{\int_{x_1} \int_{x_2} (x_1^2 - E(x_1)^2) \hat{\mathbf{f}}_i \, dx_2 \, dx_1} = R$.
5. If $x_1 = x_2$ $(R = 1)$, then $\hat{\mathbf{f}}_i = f_1 f_2 \delta(x_1 - x_2)$.
6. If x_1 is uncorrelated with x_2 $(R = 0)$, then $\hat{\mathbf{f}}_i = f_1 f_2$.

Therefore, a truncated version of representation (7.11) is a valid approximation to the probability density function of a vector of random variables. To compute the corresponding cumulative distribution function, we need to calculate a series of products of one-dimensional integrations, and for the case of a two-dimensional representation which is truncated to K number of terms:

$$\hat{\mathbf{F}}_i(\mathbf{c}) = \int_{-\infty}^{c_1} f_1 \, dx_1 \int_{-\infty}^{c_2} f_2 \, dx_2 + \sum_{k=1}^{K} R^k \int_{-\infty}^{c_1} f_1 \, q_k(x_1) \, dx_1 \int_{-\infty}^{c_2} f_2 \, q_k(x_2) \, dx_2.$$

If f_j is a Gaussian density function and $q_k(\cdot)$ is the Hermite polynomial, the result of observation 1 may be used to calculate these integrals.

Figure 7-2 shows the flow diagram of the DEMM for two different formulations of a stochastic inequality model. Since both formulations produce stochastic equality models to define the slack random variables, the use of DEMM to transform those stochastic equality models is necessary.

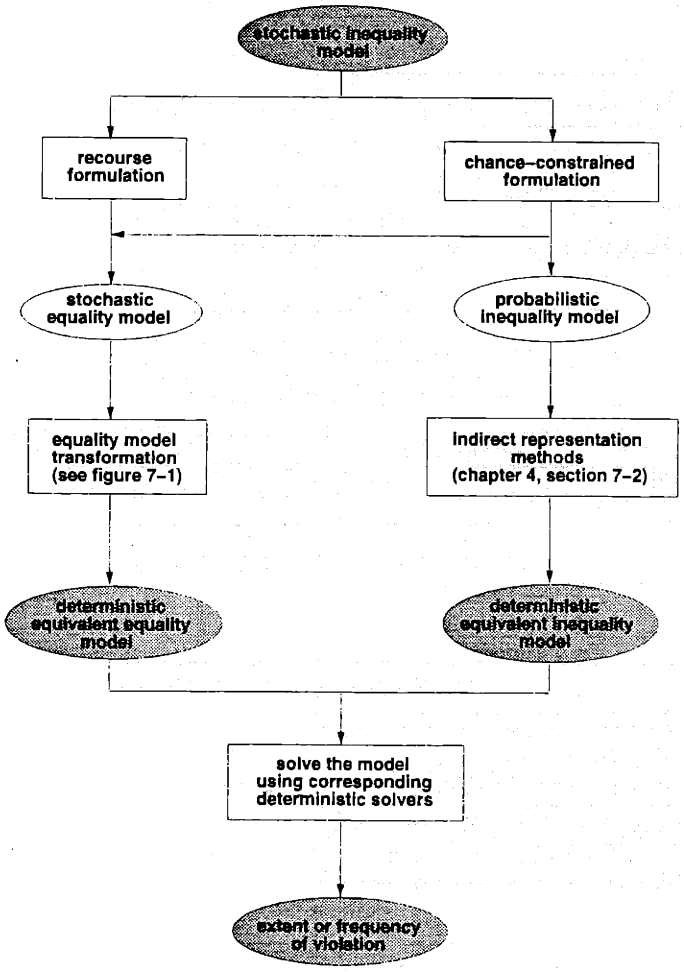


Figure 7-2: Flow diagram of the DEMM for stochastic inequality models.

7.3 Objective function transformation

In the field of stochastic programming, the formulation of a stochastic objective function becomes an integral part of the whole modeling process. Different formulations of the objective function may generate different kinds of problems. As a result, stochastic programming problems can be classified as either "here-and-now" type problems, or "wait-and-see" type problems – a division of problems derived from different formulations of the objective function. A "here-and-now" problem is a decision problem (see also chapter 6) whose objective function has the following form:

$$\max_x \mathrm{E}_z U(x, z), \tag{7.12}$$

where E_z denotes the expectation operation with respect to random variable z, and $U(\cdot)$ is the utility function under consideration. On the other hand, the objective function of the "wait-and-see" type problems can be written as:

$$\mathrm{E}_z \max_x U(x, z). \tag{7.13}$$

It should be noted that it is possible to combine both types of objective functions into one problem. As an example, the stochastic programming with recourse model which is given as the formulation (P3) in chapter 6 in fact includes these two types of problems.

With the same set of constraints, different objective functions can have different values. As an example, if we can wait and see what the values of uncertain variables in our problem are before making any decisions, or if we have perfect information on the actual values of these uncertain variables, the value of the objective function will be the highest for that set of constraints. In other words, the expected gain for having a perfect information will be maximum. On the contrary, if we have to make the decision here and now before we can observe the actual values of uncertain variables in the problem, or if we have no information at all on the uncertain variables, the value of objective function clearly will be the lowest for that set of constraints. It means that the expected gain for having no information is minimum. However, in practice, we may collect some information about the uncertain variables, and the objective function in this case can be formulated as follows:

$$\mathrm{E}_w \max_x \mathrm{E}_{z|w} U(x, z), \tag{7.14}$$

where w is the outcome of an experiment or the partial information about uncertain variables. If we have total information about uncertain variables, or $w = z$, then formulations (7.14) and (7.13) are identical. On the other hand, if we have no information about uncertain variables, or w and z are independent, then formulations (7.14) and (7.12) become identical. The following inequalities suggest the relationship between these three formulations:

$$\begin{array}{rcccl} \max_x \mathrm{E}_z U(x,z) & \leq & \mathrm{E}_w \max_x \mathrm{E}_{z|w} U(x,z) & \leq & \mathrm{E}_z \max_x U(x,z) \\ 0 & \leq & \mathrm{EVSI} & \leq & \mathrm{EVPI}, \end{array}$$

where EVSI is the expected value of sample information, and EVPI is the expected value of perfect information. These two values represent the expected gain for having partial and perfect information, and one may think of the EVSI and the EVPI as the upper bounds that

one should pay to obtain, respectively, partial information and perfect information [15].

For some cases, having the values of moments such as the mean value or the variance as the objective function is sufficient. However, for other cases, it might be more useful to deal with a probabilistic objective function. As an example, in risk analysis, one may need to minimize the probability of failure for some equipment or for some activities. Therefore, we must have a mechanism to transform not only the expected value type objective function, but also the probabilistic type objective function into their corresponding deterministic equivalences. The expected value type of objective functions can actually be transformed by applying the same procedure of the DEMM for stochastic equality models (see section 7.1). Similarly, the indirect representation methods for stochastic inequality models which are discussed in the previous section may be used to transform the probabilistic type of objective function.

7.4 Collocation method for black-box type models

As we have seen in the first section of this chapter, Galerkin's approach can produce a solution with a probabilistically-weighted error over the domain of the solution's space. However, the use of Galerkin's approach in the probabilistic variational process requires availability of the model's equations. Unfortunately, for numerous cases, we may have to deal with implicit subroutines or black-box type models. To extend the application of DEMM to such problems, we can refer to the collocation method to replace Galerkin's approach. It should be noted that if one has access to the equations of the stochastic model, the collocation method may, in fact, generate the deterministic equivalent model explicitly. This deterministic equivalent model consists of a set of deterministic problems at different values of uncertain parameters. Consequently, the collocation method produces no error in the variational process solution at those specific values of uncertain parameters.

In the notion of the probabilistic variational process, the collocation method forces the residual random variable to be deterministically zero at N specific chosen points,

$$\int_{\{\xi_i\}} f_{\{\xi_i\}}(\{\xi_i\})\, R_N(\mathbf{x},\omega)\, \delta(\{\xi_i(\omega)\} - \{p_i\}_j)\, d\{\xi_i(\omega)\} \;=\; 0;\quad j = 1,\ldots,N,$$

where $\delta(\cdot)$ denotes the delta function, and $\{p_i\}$ is the set of N collocation points.

To illustrate this idea, let us consider again the same simple problem posed in the first section,

$$a(\omega)\, x(\omega) \;=\; b(\omega),$$

where $a(\omega)$ and $b(\omega)$ are independent Gaussian random variables, and $x(\omega)$ can be represented by a polynomial chaos expansion,

$$x(\omega) \;=\; x_1 + x_2\xi_1 + x_3\xi_2 + x_4(\xi_1^2 - 1) + x_5\xi_1\xi_2 + x_6(\xi_2^2 - 1).$$

Following the notion of orthogonal collocation points [183], we may use the zeros of the third order Hermite polynomial as the basis of collocation points for this problem. Since we need to evaluate six coefficients in the polynomial chaos expansion of $x(\omega)$, six two-dimensional collocation points are chosen as follows:

$$\begin{aligned} p_1 &= (0,0) \\ p_2 &= (\sqrt{3},0) \\ p_3 &= (-\sqrt{3},0) \\ p_4 &= (0,\sqrt{3}) \\ p_5 &= (0,-\sqrt{3}) \\ p_6 &= (\sqrt{3},\sqrt{3}). \end{aligned}$$

As a result, the residual random variable becomes a deterministic variable, and for the second collocation point, we have

$$R_6(\mathbf{x},p_2) \;=\; \left(a_1 + \sqrt{3}a_2\right)\left(x_1 + \sqrt{3}x_2 + 2x_4 - x_6\right) - b_1.$$

The resulting deterministic equivalent model which based on these points can be rearranged

as follows:

$$\begin{bmatrix} a_1 & 0 & 0 & -a_1 & 0 & -a_1 \\ a_1+\sqrt{3}a_2 & \sqrt{3}a_1+3a_2 & 0 & 2a_1+2\sqrt{3}a_2 & 0 & -a_1-\sqrt{3}a_2 \\ a_1-\sqrt{3}a_2 & -\sqrt{3}a_1+3a_2 & 0 & 2a_1-2\sqrt{3}a_2 & 0 & -a_1+\sqrt{3}a_2 \\ a_1 & 0 & \sqrt{3}a_1 & -a_1 & 0 & 2a_1 \\ a_1 & 0 & -\sqrt{3}a_1 & -a_1 & 0 & 2a_1 \\ a_1+\sqrt{3}a_2 & \sqrt{3}a_1+3a_2 & \sqrt{3}a_1+3a_2 & 2a_1+2\sqrt{3}a_2 & 3a_1+3\sqrt{3}a_2 & 2a_1+2\sqrt{3}a_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} =$$

$$\begin{bmatrix} b_1 \\ b_1 \\ b_1 \\ b_1+\sqrt{3}b_3 \\ b_1-\sqrt{3}b_3 \\ b_1+\sqrt{3}b_3 \end{bmatrix}$$

and the solution to that model is given by

$$\begin{aligned} x_1 &= \frac{b_1}{a_1} \times \frac{2a_2^2 - a_1^2}{3a_2^2 - a_1^2} \\ x_2 &= \frac{b_1 a_2}{3a_2^2 - a_1^2} \\ x_3 &= \frac{b_3}{a_1} \\ x_4 &= \frac{b_1 a_2^2}{a_1(a_1^2 - 3a_2^2)} \\ x_5 &= \frac{b_3 a_2(\sqrt{3}a_2 - a_1)}{a_1(a_1^2 - 3a_2^2)} \\ x_6 &= 0. \end{aligned}$$

The result is comparable to the result using Galerkin's approach given in the first section. This example problem can also be considered as a black-box type model. The solution from this black-box type model depends on the value of ξ_1 and ξ_2,

$$x(\omega) = \frac{b_1 + b_3\xi_2}{a_1 + a_2\xi_1}.$$

Therefore, the solution using six collocation points produces the following equations which in turn can be used to define the values of coefficients of $x(\omega)$:

$$\begin{aligned} x_1 - x_4 - x_6 &= \frac{b_1}{a_1} \\ x_1 + \sqrt{3}x_2 + 2x_4 - x_6 &= \frac{b_1}{a_1 + \sqrt{3}a_2} \\ x_1 - \sqrt{3}x_2 + 2x_4 - x_6 &= \frac{b_1}{a_1 - \sqrt{3}a_2} \\ x_1 + \sqrt{3}x_3 - x_4 + 2x_6 &= \frac{b_1 + \sqrt{3}b_3}{a_1} \\ x_1 - \sqrt{3}x_3 - x_4 + 2x_6 &= \frac{b_1 - \sqrt{3}b_3}{a_1} \end{aligned}$$

$$x_1 + \sqrt{3}x_2 + \sqrt{3}x_3 + 2x_4 + 3x_5 + 2x_6 = \frac{b_1 + \sqrt{3}b_3}{a_1 + \sqrt{3}a_2},$$

The left-hand sides of these equations are derived from the polynomial chaos expansion of $x(\omega)$ at the corresponding collocation points, and the right-hand sides are obtained as the solutions to the black-box type model at the chosen collocation points. The solution for these equations is the same as the above result. It means that to generate such a result, the availability of stochastic model's equations is unnecessary when one uses the collocation method.

When the stochastic model formulation is available, one can either use Galerkin's approach or apply the collocation method to generate the deterministic equivalent model. Both methods may be regarded as procedures to construct the response surface of the model. The choice between these two methods depends on the structure of the problem and the required accuracy needed of the solution. Guidelines for selecting the method of the deterministic variational process [183] can also be applied to the probabilistic variational process. As an example, the moments values of the response variables or solutions may be more accurately computed by using Galerkin's approach. On the contrary, the probability that response variables are exceeding some constant values may be more precisely calculated by using the collocation method, especially when the chosen points produce solutions close to these constant values.

The requirement to compute multi-dimensional integrations in Galerkin's approach limits the choice of direct representations for random variables. We need to use the polynomial chaos expansion which consists of Hermite polynomials and Gaussian random variables for replacing a problem of symbolic integrations with a problem of symbolic substitutions. We also have to use polynomial approximations for representing other non-linear forms. To remove such limitations, one can use the collocation method. By using such a method, one can now deal with non-linear models of non-polynomial form and can use problem-specific polynomial chaos expansions which are based on the specific probability density functions of uncertain parameters in the model. A package of routines for generating orthogonal polynomials relative to arbitrary weight functions has been developed by Gautschi [52].

For a general nonlinear case, the optimal collocation method can be identified with a Galerkin method where the integrals are evaluated by optimal quadrature formulas [183]. Therefore, the result of the collocation method can be considered as an approximation to the result of Galerkin's approach. Nevertheless, the use of collocation method surely extends the area of application of DEMM. The flow diagram of the DEMM with the collocation method for stochastic models of explicit and black-box types is shown in figure 7-3.

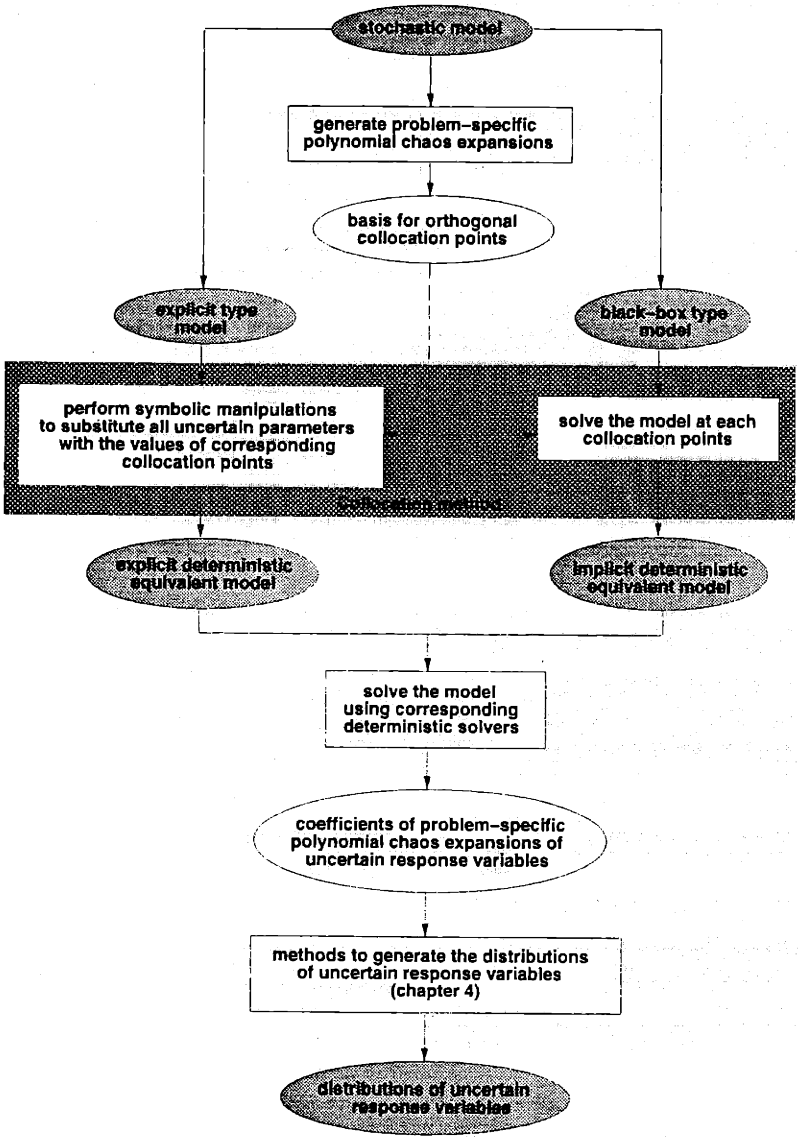


Figure 7-3: Flow diagram of the DEMM with the collocation method for stochastic models.

Chapter 8

Implementation

The development of a language based on the procedures described in the previous chapter for the deterministic equivalent modeling method becomes necessary when we have to deal with large and complex stochastic models. This language will automate the transformation process of a stochastic model into its deterministic equivalence. Such an automation requires the application of reliable symbolic manipulation routines as well as dedicated lexical scanner and parser generators. Therefore, the first section of this chapter will highlight issues related to the use of symbolic manipulation routines. It is followed by a section that discusses the language of the deterministic equivalent modeling method, and in the third section, we will discuss the reuse of deterministic solvers for solving stochastic models. The last section describes the structure of an input file written in the prototype languange of the deterministic equivalent modeling method.

8.1 Symbolic Manipulations Kernel

It is mentioned in chapter 7 that the orthogonalization process of the Galerkin's approach requires the use of the symbolic multi-dimensional integration routine. However, if the stochastic model of interest is of polynomial form, the special structure of polynomial chaos expansion can replace the symbolic integration process with symbolic substitution process. For such a case, the need of computer algebra systems may be unnecessary since we only need to perform symbolic manipulations. By its definition [181], computer algebra programs provide the user a control over massive amounts of built in knowledge of mathematics, and much of this is used automatically. On the other hand, the symbolic manipulation program is mainly a rewrite system, which is a set of directed of equations used to compute by repeatedly replacing subterms of a given formula with equal terms until the simplest form possible is obtained [36]. This theory of rewriting is to some extent an outgrowth of the study of Church's λ-calculus and Curry's Combinatory Logic (for further references on these topics, please see [134]). Clearly, for our problems, the symbolic manipulation program is sufficient and appropriate for performing the substitution process of polynomials of Gaussian random variables in the stochastic models with their corresponding moment values. As an example, a simple rewrite rule can be applied to substitute moments values of a standard Gaussian random variable ξ,

$$\xi^n \rightarrow E_n,$$

where E_n is the n-th element of set $E = \{0, 1, 0, 3, 0, 3 \times 5, 0, 3 \times 5 \times 7, 0, 3 \times 5 \times 7 \times 9, \ldots\}$. Furthermore, if we have M independent standard Gaussian random variables, we can use the following rewrite rules,

$$\begin{aligned} \xi_1^n &\rightarrow E_n \\ \xi_2^n &\rightarrow E_n \\ &\vdots \\ \xi_M^n &\rightarrow E_n. \end{aligned}$$

The transformation of stochastic models requires a symbolic manipulation kernel that capable for handling such simple rewrite rules. To cope with practical applications or problems, another critical requirement for that kernel is the ability for dealing with large algebraic formulae. From a review of available computer algebra systems [156], we choose to use a public domain symbolic manipulation program, called **FORM** version 1, which satisfies both requirements [181]. As a result, the above rewrite rules can be written in **FORM** statements as follows:

```
id      x1^n? = Expected[n];
id      x2^n? = Expected[n];
              .
              .
              .
id      xM^n? = Expected[n];
id      Expected[1] = 0;
id      Expected[2] = 1;
id      Expected[3] = 0;
id      Expected[4] = 3;
id      Expected[5] = 0;
id      Expected[6] = 15;
                    .
                    .
                    .
```

These rewrite rules use the "id" statements [180]. "id" stands for identity and this statement is the most important statement in symbolic manipulation. It allows the user to compose complicated operations by successive substitutions. For example, to calculate the probabilistic orthogonalization of residual,

$$R(\xi) = a_1 + a_2\xi_1 + a_3\xi_2 + a_4\xi_1^4 + a_5\xi_1^5,$$

with respect to the basis function,

$$H_1(\xi) = \xi_1,$$

we may use the following set of **FORM** statements

```
#-

Symbols  n,x1,x2;
```

```
Set      Gaussian:x1,x2;
Symbols  expect1,expect2,expect3,expect4,expect5,expect6;
Set      expected:expect1,expect2,expect3,expect4,expect5,expect6;
Symbols  a1,a2,a3,a4,a5;

Global   residual = a1 + a2*x1 + a3*x2 + a4*x1^4 + a5*x1^5;

.sort

Local    orthogonal = x1*residual;

Skip     residual;
print    orthogonal;

id       x1?Gaussian^n? = expected[n];
id       x2?Gaussian^n? = expected[n];
id       expect1 = 0;
id       expect2 = 1;
id       expect3 = 0;
id       expect4 = 3;
id       expect5 = 0;
id       expect6 = 15;

.end
```

to obtain the result of

$$\int_{\infty}^{\infty} f_{\xi_1,\xi_2}(\xi_1,\xi_2)\; R(\xi)\; H_1(\xi)\; d\xi,$$

```
   FORM by J.Vermaseren. Version 1.0 22-dec-1990
   #-

Time =       0.00 sec    Generated terms =          5
         residual        Terms in output =          5
                         Bytes used      =         98

Time =       0.02 sec    Generated terms =          2
       orthogonal        Terms in output =          2
                         Bytes used      =         34

   orthogonal =
      a2 + 15*a5;
```

which equals to $a_2 + 15a_5$.

In practice, we need to generate such an input file to program **FORM** automatically for a given stochastic model. We, therefore, have to develop a program that can transform the description of a stochastic model into a corresponding input file for **FORM** whose result or output is the description of deterministic equivalent model. Figure 8-1 shows

the flow diagram of the transformation process of a stochastic model into its deterministic equivalence. There are two compilers needed in this process. One is to transform the

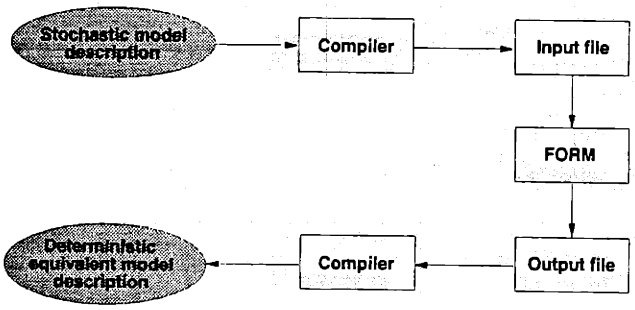


Figure 8-1: The flow of information in the deterministic equivalent modeling method

stochastic model description into an input file of **FORM**. In other words, this compiler defines a language or an environment where the description of the stochastic model can be written, and it will be discussed in the next section. The other one is to transform the output file of **FORM** into the description of deterministic equivalent model. Consequently, this second compiler should be able to generate the equivalent model in the programming language or the environment where the deterministic solvers work.

8.2 Deterministic equivalent modeling language

The main objective of the deterministic equivalent modeling language is to describe the stochastic model of interest. Since the above example of input file to **FORM**, in fact, suggests the required structure of deterministic equivalent modeling language, the description of a stochastic model should therefore include: the symbols for standard independent Gaussian random variables in the polynomial chaos expansions; the moments values for these random variables; the forms of expansions for random parameters and random variables in the model; the declaration of the model itself; and the orthogonalization process. We will describe each of these items in the following.

To set the symbols for Gaussian random variables for polynomial chaos expansions, we use the keywords `Set [BF]`, in this case, `BF` stands for "Basis Function" elements. As an example, if we want to set symbols `x1` and `x2` as two independent standard Gaussian random variables, we can use the following statement

```
Set [BF] Gaussian = {x1,x2};
```

`Gaussian` is an arbitrary name for referring to both `x1` and `x2`. We also see from this simple statement that we use a semi colon, ";", to end a statement.

Similarly, keywords `Set [MV]`, where `MV` represents "Moments Values", are used to set the values of moments of the corresponding random variables declared in the `Set [BF]` statement. For example, we can set the values of the first sixth moments of those two Gaussian random variables `x1` and `x2` with the statement

```
Set [MV] expected -> Gaussian = {0,1,0,3,0,15};
```

The right arrow, "`->`", in this statement denotes the word "of", or the word "for". It means that the symbols `expected -> Gaussian` may be read as "the expected of Gaussian", or "the expected for Gaussian".

Furthermore, if x1 and x2 represent Gaussian random variables, and x3 and x4 denote the exponential random variables, we may use the following statements to describe them and their first four moments values

```
Set [BF] Gauss = {x1,x2},
         expo  = {x3,x4};

Set [MV] expectG -> Gauss = {0,1,0,3},
         expectE -> expo  = {1,1,2,9};
```

In this example, we use a comma, ",", for extending any setting environments such that they can have more than one statement.

The form of polynomial chaos expansion can be set by using the keywords `Set [PC]`. Clearly, PC in here stands for the "Polynomial Chaos" expansions. As an example, to set the form of a six-term polynomial chaos expansion that has the first and the second order of two dimensional Hermite polynomials, we use a statement such as

```
Set [PC] polychaosH = {1,x1,x2,x1*x2,x1^2-1,x2^2-1};
```

Such a setting statement will be useful in the case that many response variables or parameters are represented by the same form of polynomial chaos expansion.

After setting the form of the polynomial chaos expansion, we can now define the structures of random parameters and random variables in the stochastic model. The setting environment for random parameters begins with keywords `Set [RP]`. There are several options for declaring the structure of random parameters. The first option allows us to write explicitly the form of expansion for a random parameter. As an example, if the parameter p is a Gaussian random variable with mean value p0 and standard deviation p1, we may declare it as follows:

```
Set [RP] p = p0 + p1*x1;
```

where x1 has already been declared as a standard Gaussian random variable. In the second option, we can use the previous setting of a polynomial chaos expansion to generate the expansion for a random parameter,

```
Set [RP] p -> polychaosH;
```

This statement will generate six-term polynomial chaos expansion based on the form of polychaosH. It means that the parameter p will have six coefficients, and they are stored as an array by the name p. As an example, if we use the C programming language, we will have an array `p[1]` to `p[6]` which represents those six coefficients. Another option provides us with alternative names for that array. Therefore, if we want to store the first three coefficients of p by an array whose name is pother, and it runs from index 2 to 4, and the rest of coefficients are called as pt1, pt3, and pt4, we may use the following statement:

```
Set [RP] p -> polychaosH = {{pother[2,...,4]},pt1,pt3,pt4};
```

Those three options are also available in defining the structure of random variables. However, we use the keywords `Set [RV]` to begin setting the structure of random variables. It should be pointed out that we actually can use `Set [RP]` to set the random variables, but the set command `Set [RV]` will make the definition of a stochastic model clearer. The use of both setting keywords will definitely help in reading the stochastic model, and will

make clear which symbols denote uncertain parameters, and what the names of uncertain variables are. Therefore, to define a random variable y that has polynomial chaos expansion of the form `polychaosH`, we write the statement:

```
Set [RV] y -> polychaosH;
```

After all uncertain parameters and variables have been defined, the next step is to write the stochastic models that relate those quantities. The model environment begins with the keyword `Model` or `Models`. In the deterministic equivalent modeling language, we define the stochastic equality model by both its expression and the trial functions used for the orthogonalization process. As an example, we can use the following statement:

```
Model ortho -> polychaosH := y^2 + y;
```

to define an equality model $y^2 + y = 0$, to use each element of set `polychaosH` as the trial function in the orthogonalization process, and to put the results of orthogonalization process in an array with name `ortho`. Therefore, in this case, the first trial function equals to 1, and the result of the orthogonalization process when we use the **C** programming language is

```
ortho[1] = pow(y[1],1.0*2) + y[1] + pow(y[2],1.0*2) +
           2*pow(y[3],1.0*2) + 2*pow(y[4],1.0*2) +
           2*pow(y[5],1.0*2) + 6*pow(y[6],1.0*2);
```

Similar to the case of statements for defining random parameters or variables, we have an option to choose specific names for the results of the orthogonalization process. As an example, we can use an array with name `orthox` from index 2 to 4, and `orth1`, `orth5`, and `orth3` to represent the result,

```
Model ortho := y^2 + y =
      {{orthox[2,...,4],orth1,orth5,orth3} -> {polychaosH[1,...,6]}};
```

All the above setting and model environments are used to describe stochastic models and their components in the language of deterministic equivalent modeling method. The full syntax of the deterministic equivalent modeling language can be seen in appendix E. To implement such a syntax, we use **Flex** as the lexical scanner generator [114], and **Bison** as the parser generator [41]. The input files for these generators and other auxiliary routines are given in appendix E.

8.3 Reuse of Deterministic Solvers

The previous section has discussed the language of the deterministic equivalent modeling method, however, it did not mention how to reuse deterministic solvers to solve the stochastic models. In the `Models` environment, we define the expressions of stochastic models and the names of variables to store the deterministic equivalent models generated from the orthogonalization process. Therefore, to reuse the deterministic solvers for solving those deterministic equivalent models, we need to transform the results of **FORM** into the corresponding language where the deterministic solvers work.

The keyword `Format` is implemented in the language of the deterministic equivalent modeling method to allow the user to choose the right format for the deterministic equivalent models. Currently, there are four choices available which include: the **C** and the

Fortran programming languages, or the **GAMS** [21] and the **Abacuss** [10] environments. As an example, to make the deterministic equivalent model written in the **C** programming language, we use the following statement:

```
%Format C
```

We see that before the keyword `Format`, we put the percent sign `%`. This percent sign denotes the command line in the deterministic equivalent modeling language, and it should be placed in the beginning of a line. Furthermore, there should be no space between the percent sign and the keyword `Format`.

Other keywords related to the process of generating deterministic equivalent model are `Start` and `End`. Every stochastic model described in the `Model` environment will be transformed into the corresponding deterministic equivalent model. This generated equivalent model will be stored in a file whose name is defined with the `Start` keyword, and all deterministic equivalent models of stochastic models that are declared between the `Start` and `End` keywords will be stored in the same file. As an example, the following statements make the deterministic equivalent models of stochastic models `stoch1` and `stoch2` to be placed in the file `deter1and2` whose extension depends on the `Format` statement.

```
Start deter1and2

Models stoch1 -> polychaosH := y^2 + y,
       stoch2 -> polychaosH := 2*y - 3*y^3;

End
```

In other words, if the `Format` statement chooses the **C** programming language, the deterministic equivalent models will be written in the format of **C** language in the file `deter1and2.c`. Between the `Start` and `End` keywords, we can put local definitions or declarations of random variables, moments values, random parameters and variables. Such a feature is similar to the feature of local variables in a procedure or a function in the programming language. In this case, we use a single symbol table in which each symbol or variable in different `Start` and `End` regions has different scope numbers [47]. Consequently, we write the global definitions or declarations outside any `Start` and `End` regions.

Since we can set the outputs of **FORM** to be written in the **Fortran** format (however, they are not in the right column), we only need to implement several translators from **Fortran** to **C**, **GAMS**, and **Abacuss**. We also have to write a script file to translate the outputs of **FORM** into the right position for **Fortran**. The translators are written with the help of **Flex** and **Bison**, the scanner and parser program generators. Input files to these generators and auxiliary files such as script files can be seen in the appendix E. These script files run in the Unix environment, and require Unix commands **sed** and **awk** [80].

8.4 Structure of Input File

To illustrate the structure of a typical input file written in the prototype language of the deterministic equivalent modeling method. We again consider a simple problem described in chapter 7. A linear stochastic algebraic equation,

$$a(\omega)\, x(\omega) \;=\; b(\omega),$$

where $a(\omega)$ and $b(\omega)$ are independent Gaussian random variables whose mean values are denoted by a_1 and b_1, and their standard deviations are represented by a_2 and b_2 respectively. In other words, the polynomial chaos expansions for these two random variables are written as follows:

$$\begin{aligned} a(\omega) &= a_1 + a_2\xi_1(\omega) \\ b(\omega) &= b_1 + b_2\xi_2(\omega), \end{aligned}$$

where $\xi_i(\omega)$ is a standard Gaussian random variable. Similarly, the response variable $x(\omega)$ is also expanded into a polynomial chaos expansion. In this case, we use a six-term polynomial chaos expansions,

$$x(\omega) = x_1 + x_2\xi_1 + x_3\xi_2 + x_4(\xi_1^2 - 1) + x_5\xi_1\xi_2 + x_6(\xi_2^2 - 1)$$

to approximate random variable $x(\omega)$. The resulting deterministic equivalent equations for this case are in the following form (see chapter 7),

$$\begin{bmatrix} a_1 & a_2 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & 0 & 2a_2 & 0 & 0 \\ 0 & 0 & a_1 & 0 & a_2 & 0 \\ 0 & 2a_2 & 0 & 2a_1 & 0 & 0 \\ 0 & 0 & a_2 & 0 & a_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} b_1 \\ 0 \\ b_3 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{8.1}$$

Now, we want to use the prototype language of the deterministic equivalent modeling method to generate those deterministic equivalent equations. Before we apply that language to the stochastic problem, let us consider how to solve the corresponding deterministic problem. A set of deterministic linear algebraic equations can be solved by using the LU decomposition method. As an example, we may write the following program in **C** language to solve the simple deterministic problem $ax = b$.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "utility.c"
#include "ludcmp.c"
#include "lubksb.c"

double a,b;

void coeff(double **c)
{
     c[1][1] = a;
}

void force(double *f)
{
     f[1] = b;
}

#define N 1
```

```
int main(void)
{
  int i,*indx;
  double d;
  double **c,*f;
  a=2.0;
  b=1.0;
  indx=ivector(1,N);
  c=dmatrix(1,N,1,N);
  f=dvector(1,N);
  coeff(c);
  force(f);
  ludcmp(c,N,indx,&d);
  lubksb(c,N,indx,f);
  printf("i\tx\n");
  for(i=1;i<=N;i++)
    { printf("%d\t%lf\n",i,f[i]); }
  return (0);
}
```

Even though the problem can be simply solved as $x = \frac{b}{a}$, we use the LU solver to show that the same deterministic solver will be used to solve the stochastic model. If a and b are random variables, then the content of procedures `coeff` and `force` as well as the definitions and declarations of a and b should be changed accordingly. In this case, a and b will be vectors, each with two elements to represent their mean values and standard deviations. Therefore the C statements

```
      .
      .
      .
double a,b;
      .
      .
      .
    a=2.0;
    b=1.0;
      .
      .
      .
```

become

```
      .
      .
      .
double *a,*b;
      .
      .
      .
    a=dvector(1,2);
    b=dvector(1,2);
    a[1]=2.0;a[2]=0.2;
```

```
b[1]=1.0;b[2]=0.1;
 .
 .
 .
```

in the case that the mean values of a and b are 2.0 and 1.0, and their standard deviations equal to 0.2 and 0.1 respectively. Furthermore, the value of N is now equal to 6 because we will have six unknowns,

```
#define N 6
```

Therefore, this problem's input file to the deterministic equivalent modeling method compiler is in the following form.

```
%{
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "utility.c"
#include "ludcmp.c"
#include "lubksb.c"

double *a,*b;
%}

# use c language for output files
%Format c

%%

# two independent Gaussian random variables,
# e1 and e2, are used as coordinates
Set [BF] basis = {e1,e2};

# define the value of moments for those coordinates
# for example the fourth moment is 3
Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135};

# we use three terms for x
Set [PC] pcx = {1,e1,e2,e1^2-1,e1*e2,e2^2-1};

# calculate coefficient matrix
%{
void coeff(double **c)
{
%}

# and put it into coeff.c
Start coeff

# we use two terms for a
Set [PC] pca = {1,e1};

Set [RP] a -> pca;
```

```
Models   coef1 := a =
          {{c[1][1,...,6]} -> {pcx[1,...,6]}},
         coef2 := a*e1 =
          {{c[2][1,...,6]} -> {pcx[1,...,6]}},
         coef3 := a*e2 =
          {{c[3][1,...,6]} -> {pcx[1,...,6]}},
         coef4 := a*(e1^2-1) =
          {{c[4][1,...,6]} -> {pcx[1,...,6]}},
         coef5 := a*e1*e2 =
          {{c[5][1,...,6]} -> {pcx[1,...,6]}},
         coef6 := a*(e2^2-1) =
          {{c[6][1,...,6]} -> {pcx[1,...,6]}};

End

%{
}
%}

# calculate force vector
%{
void force(double *f)
{
%}

# and put it into force.c
Start force

# we use two terms for b
Set [PC] pcb = {1,e2};

Set [RP] b -> pcb;

Model    force := b =
          {{f[1,...,6]} -> {pcx[1,...,6]}};

End

%{
}
%}

%%

#define N 6

int main(void)
{
  int i,*indx;
  double d;
  double **c,*f;
  a=dvector(1,2);
  b=dvector(1,2);
  a[1]=2.0;a[2]=0.2;
  b[1]=1.0;b[2]=0.1;
  indx=ivector(1,N);
  c=dmatrix(1,N,1,N);
  f=dvector(1,N);
```

```
  coeff(c);
  force(f);
  ludcmp(c,N,indx,&d);
  lubksb(c,N,indx,f);
  printf("i\tx\n");
  for(i=1;i<=N;i++)
    { printf("%d\t%lf\n",i,f[i]); }
  return (0);
}
```

There are several symbols which appear in this input file but have not been explained previously. Any lines between command lines %{ and %} are copied verbatimly to the output of the deterministic equivalent modeling method compiler. Therefore the output from giving this input file to the DEMM compiler will actually contain these lines:

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "utility.c"
#include "ludcmp.c"
#include "lubksb.c"

double *a,*b;

%{
void coeff(double **c)
{
   .
   .
   .
}

void force(double *f)
{
   .
   .
   .
}

#define N 6

int main(void)
{
  int i,*indx;
  double d;
  double **c,*f;
  a=dvector(1,2);
  b=dvector(1,2);
  a[1]=2.0;a[2]=0.2;
  b[1]=1.0;b[2]=0.1;
  indx=ivector(1,N);
  c=dmatrix(1,N,1,N);
  f=dvector(1,N);
  coeff(c);
  force(f);
```

```
    ludcmp(c,N,indx,&d);
    lubksb(c,N,indx,f);
    printf("i\tx\n");
    for(i=1;i<=N;i++)
      { printf("%d\t%lf\n",i,f[i]); }
    return (0);
}
```

By evaluating the above output's lines, one may ask why the lines from `#define N 6` to the rest of input file are also included even though they are not contained between the command lines %{ and %}. To answer that question, we should see the structure of a typical input file. A typical input file to the DEMM compiler should have one `Format` command and a pair of percent sign commands:

```
      .
      .
      .
%Format C
      .
      .
      .
%%
      .
      .
      .
%%
      .
      .
      .
```

Between those pair of percent sign commands, we can use the prototype language of the deterministic equivalent modeling method described in section 8.2. Any lines after the second percent sign command are copied verbatimly to the output of the DEMM compiler. Another symbol that has not been explained is the pound sign #. Any part of the line after this sign will be treated as comments.

To learn more about the structure of input files, we may see input files of examples described in the next chapter, which are given in the appendix F. It should be noted that the deterministic equivalent modeling method language is in the development phase, and it will be continuously changed and improved to meet the needs arised from various practical problems.

Chapter 9

Uncertainty Analysis Examples

Examples of stochastic models in three main areas of chemical and environmental engineering systems are considered in this chapter. In the first area, three steady-state examples which include: a simple algebraic equations model, the climate forcing problem, and a simple process flowsheet simulation example, will be discussed. The application of the collocation method to the flowsheet simulation demonstrates the value and practicality of this method to perform the uncertainty analysis of the black-box type models. These steady-state examples are followed by three dynamical system problems. A mass action kinetics model of which the values of uncertain parameters are close the bifurcation point becomes the first example of the dynamical systems. The second one describes the problem in the photochemical smog mechanism where the predicted concentration of O_3 is actually very sensitive to one of two assumed reaction rate constants. Reduced H_2/O_2 mechanism becomes the third dynamical system example.

The second area consists of inequality models. In this area, we will see two examples that show the use of Edgeworth series expansion to calculate the value of stochastic flexibility index of the system. These two examples cover linear and nonlinear inequality models. The last area illustrates stochastic optimization problems. In the first example of stochastic optimization models, the trade-off between the flexibility of a sequence of reactors and the expected product concentration is analyzed. The other example describes the application of the collocation method to the generation of the probability density function of the optimal value. All these examples show the ease of applying the deterministic equivalent modeling method to a wide range of stochastic models, as well as the accuracy of this method. Table 9.1 presents the important points from these examples.

9.1 Algebraic and differential equations examples

9.1.1 Simple algebraic equations model

A simple steady state model of a fiber process is described by the following three equations:

$$\begin{aligned} x &= \frac{c_{11}u - c_{12}uv}{1 + c_{13}v} \\ y &= \frac{c_{21}u - c_{22}uv}{1 + c_{23}v} \\ z &= \frac{c_{31}u - c_{32}uv}{1 + c_{33}v}, \end{aligned}$$

Example	Methods	Key points
9.1.1 Algebraic equations	Analytical and Galerkin	– Variability of outputs can be larger than variability of each input
9.1.2 Algebraic equations	Analytical and Galerkin	– Concept of uncertainty factors
9.1.3 Algebraic equations	Galerkin and Collocation	– DEMM can be used in either equation or modular-based process simulator
9.1.4 Differential equations	Galerkin	– Uncertainty may change the equilibrium positions
9.1.5 Differential equations	Galerkin	– Propagation of uncertainty in reaction rate constants to the prediction of species concentrations
9.1.6 Differential equations	Galerkin	– Sensitivity × standard deviation measure
9.2.1 Inequality model	Edgeworth expansion	– Stochastic flexibility index – Linear problem
9.2.2 Inequality model	Edgeworth expansion	– The use of Adomian polynomials for approximating non-polynomials
9.3.1 Optimization model	Gallerkin	– Trade-off problem – Chance-constrained formulation
9.3.2 Optimization model	Collocation	– Problem specific polynomial chaos expansion – Distribution problem

Table 9.1: Key points of each example.

where the matrix of constants c_{ij} is given as follows:

$$\{c_{ij}\} = \begin{bmatrix} 2.26465 & 0.0226465 & 0.00727279 \\ 12.7992 & 0.145265 & 0.00727279 \\ 1.71111 & 0.0171111 & 0.00727279 \end{bmatrix}.$$

The fundamental issue in this process is a question of over control. These three equations are used to calculate the value of x, y, and z from inputs u and v. In this problem, we would like to see whether by going through these calculations and adjusting all of the time may be causing more variability then just assuming an average value for u and v and then assuming specific values for x, y, and z. Furthermore, we may assume for this problem that u and v are normally independent distributed with mean values 155.5 and 85.175, and standard deviations 0.3 and 0.06 respectively. The question is simply; given the information of the variability in the inputs, what is the variance of x, y, and z?

Compared to the mean values of u and v, the magnitude of their standard deviations is small. This might suggest that with such a structure of model, the distribution of x, y, and z may be, in fact, very close to Gaussian. To verify this statement, we will, in the following, develop the analytical distribution for the response variables of this model by using formulas that are given in the Papoulis' book [120].

Those three equations in the model are all of a similar form, consequently, one may only have to derive the analytical distribution for one response variable. Since we can assume that u and v are Gaussian with mean values u_0, and v_0, and standard deviations u_1, and

v_1 respectively, the equation for x becomes

$$x = \frac{k + l\xi_1 + m\xi_2 + n\xi_1\xi_2}{p + q\xi_2}$$

where ξ_1 and ξ_2 are independent standard Gaussian random variables (mean value = 0, and variance = 1), and the constants are defined by

$$\begin{aligned} k &= c_{11}u_0 - c_{12}u_0v_0 \\ l &= c_{11}u_1 - c_{12}u_1v_0 \\ m &= -c_{12}u_0v_1 \\ n &= -c_{12}u_1v_1 \\ p &= 1 + c_{13}v_0 \\ q &= c_{13}v_1. \end{aligned}$$

The analytical distribution of x can be derived as follows:

$$f_x(x) = \int_{-\infty}^{\infty} f_{xt}(x,t)dt$$

where

$$\begin{aligned} f_{xt}(x,t) &= \left|\frac{p+qs}{l+ns}\right| f_{e_1e_2}(r,s) \\ r &= \frac{k - px + mt - qxt}{-l - nt} \\ s &= t \\ f_{e_1e_2}(r,s) &= \frac{1}{2\pi}\exp(-\frac{r^2+s^2}{2}). \end{aligned}$$

Figure 9-1 shows the analytical probability density function of x. Clearly, the shape of that density function is quite close to that of Gaussian.

To use DEMM for solving this problem (the DEMM input file for this example can be seen in the appendix F), we treat the model of x as a linear algebraic equation problem:

$$(p + q\xi_2)\, x = k + l\xi_1 + m\xi_2 + n\xi_1\xi_2$$

In this case, we apply a six-term polynomial chaos expansion to approximate the response variable x,

$$x = x_0 + x_1\xi_1 + x_2\xi_2 + x_3\xi_1\xi_2 + x_4(\xi_1^2 - 1) + x_5(\xi_2^2 - 1)$$

Substituting this representation back to the linear model, and applying the Galerkin's ap-

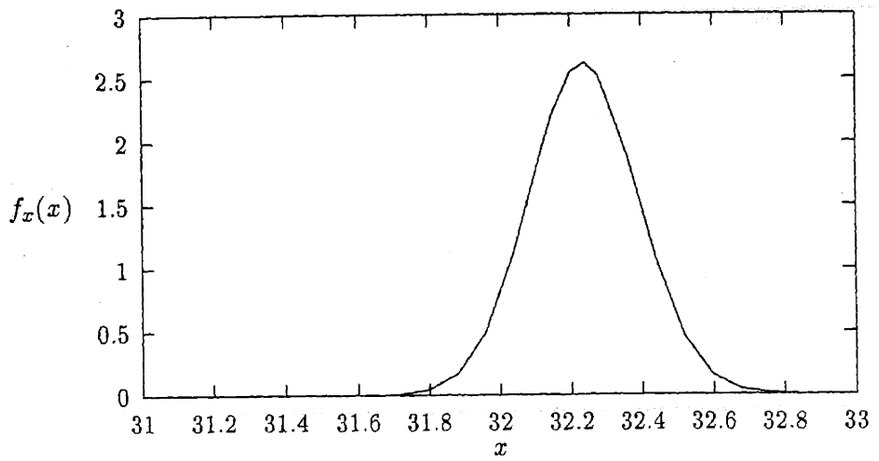


Figure 9-1: Analytically derived probability density function of x.

proach in the variational process, we produce the following matrix equation:

$$\begin{bmatrix} p & 0 & q & 0 & 0 & 0 \\ 0 & p & 0 & q & 0 & 0 \\ q & 0 & p & 0 & 0 & 2q \\ 0 & q & 0 & p & 0 & 0 \\ 0 & 0 & 0 & 0 & 2p & 0 \\ 0 & 0 & 2q & 0 & 0 & 2p \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} k \\ l \\ m \\ n \\ 0 \\ 0 \end{bmatrix}. \tag{9.1}$$

Any standard deterministic linear algebraic equations solver can solve matrix equation (9.1), and for this problem we apply the LU decomposition method [127]. The result is given by:

$$\begin{aligned} x \;=\; & 32.23707947 + 0.06219364\,\xi_1 - 0.13915704\,\xi_2 - 0.00026847\,\xi_1\,\xi_2 + \\ & 0.0\,(\xi_1^2 - 1) + 0.00003750\,(\xi_2^2 - 1). \end{aligned}$$

From this result, we can also compute the mean value, variance, coefficient of skewness and other measures of distribution. As an example, the mean value of x is 32.23707947, its variance equals to 0.0232328, and the coefficients of skewness and kurtosis are 0.00516721, and 3.00005 respectively. The last two values imply that the probability density function of x is quite close to the Gaussian density function since the coefficients of skewness and kurtosis for a Gaussian random variable are 0 and 3 respectively. It also means that we may approximate the density function of x with the density function of a Gaussian random variable whose mean value and standard deviation are the same as those of x. Figure 9-2 shows the Gaussian approximation of the probability density function of x. We can see that there is a close agreement between the analytically derived probability density function and the probability density function from the result of DEMM.

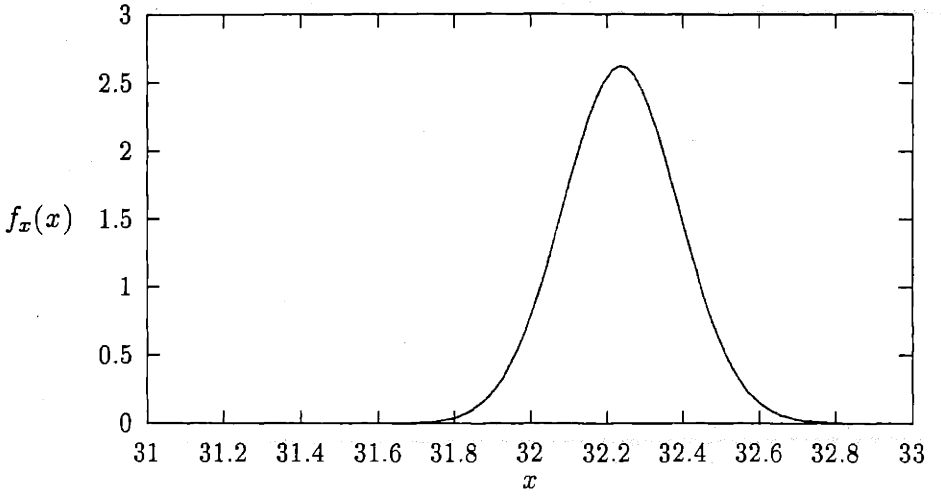


Figure 9-2: Probability density function of x derived from the DEMM result.

It is clear that this approach can be applied to solve the other response variables, y and z. We only need to change the values of coefficients and forcing factors in the matrix equation (9.1). The results for y and z are given by

$$\begin{aligned} y &= 40.93109432 + 0.07896689\,\xi_1 - 0.84792537\,\xi_2 - 0.00163587\,\xi_1\,\xi_2 + \\ &\quad 0.0\,(\xi_1^2 - 1) + 0.00022848\,(\xi_2^2 - 1) \\ z &= 24.35753021 + 0.04699208\,\xi_1 - 0.10514361\,\xi_2 - 0.00020285\,\xi_1\,\xi_2 + \\ &\quad 0.0\,(\xi_1^2 - 1) + 0.00002833\,(\xi_2^2 - 1). \end{aligned}$$

Similar to the result of response variable x, we can also calculate the variances and higher moments values of y and z from these DEMM results. As an example, for the case of y, its variance equals to 0.725216, its coefficient of skewness is 0.00266008, and its coefficient of kurtosis equals to 3.00005. The variance of z equals to 0.0132635, its coefficient of skewness is 0.00516704, and its coefficient of kurtosis becomes 3.00004.

All these results imply two things. The first one is that for a given variability in the inputs, the variability of the outputs of the model is quite noticeable. Other suggests that both the structure of the model and the values of constants in the model dictate the shape of distribution function of the outputs to be close to a Gaussian distribution.

9.1.2 Climate forcing by anthropogenic aerosols

To show the notion and use of multiplicative uncertainty factor for positive-valued uncertain parameters, we will consider a simple box model for computing direct climate forcing by sulfate aerosol [29]. This model describes the areal mean shortwave forcing ΔF_R resulting from an increase in sulfate aerosol concentration,

$$\Delta F_R = -\frac{1}{2} F_T T^2 (1 - A_c)(1 - R_s)^2 \beta \delta_{SO_4^{2-}}, \tag{9.2}$$

where F_T is the top-of-the-atmosphere radiative flux, T is the fraction of incident light that is transmitted by the atmospheric layer above the aerosol layer, A_c is the fractional cloud cover, R_s is the mean albedo of the underlying surface, β is the fraction of the radiation scattered upward by the aerosol, and $\delta_{SO_4^{2-}}$ is the areal mean optical depth of the aerosol which given by the following equation

$$\delta_{SO_4^{2-}} = \alpha_{SO_4^{2-}} f(RH) B_{SO_4^{2-}}, \tag{9.3}$$

and $\alpha_{SO_4^{2-}}$ is the molar scattering cross section of sulfate at a reference low relative humidity (30%), $f(RH)$ denotes the relative increase in scattering due to the increase in particle size associated with hygroscopic accretion of water with increasing relative humidity, and $B_{SO_4^{2-}}$ is the areal mean column burden of sulfate aerosol and it is related to parameters involving sources and removal of atmospheric sulfate:

$$B_{SO_4^{2-}} = 3 \frac{Q_{SO_2} Y_{SO_4^{2-}} \tau_{SO_4^{2-}}}{A}, \tag{9.4}$$

where Q_{SO_2} is the source strength of anthropogenic SO_2, $Y_{SO_4^{2-}}$ represents the fractional yield of emitted SO_2 that reacts to produce SO_4^{2-} aerosol, $\tau_{SO_4^{2-}}$ denotes the sulfate lifetime in the atmosphere, and A is the area of the region to where the material is presumed to be confined. The values of uncertain parameters in the above box model, equations (9.2 - 9.4), are given in table 9.2 below [29, 123]. Before we solve this stochastic model of direct

Number	Parameter	Central value	Uncertainty factor
1	Q_{SO_2}	71×10^{12}	1.15
2	$Y_{SO_4^{2-}}$	0.5	1.5
3	$\tau_{SO_4^{2-}}$	0.0152778	1.5
4	$\alpha_{SO_4^{2-}}$	5.0	1.4
5	$f(RH)$	1.7	1.2
6	T^2	0.58	1.4
7	$1 - A_c$	0.39	1.1
8	$(1 - R_s)^2$	0.72	1.2
9	β	0.3	1.3
10	F_T	1370.0	1.0
11	A	5×10^{14}	1.0

Table 9.2: Values of parameters in the box model.

climate forcing, we need to establish the notion of multiplicative uncertainty factors, and a connection between log-normal and normal random variables as well.

A Gaussian uncertain parameter, x, is naturally represented by its mean value and its standard deviation which is commonly treated as the additive uncertainty factor,

$$x = \mu(x) \pm \sigma(x),$$

where $\mu(x)$ and $\sigma(x)$ are the mean value and standard deviation of x respectively. Similarly, a log-normal uncertain parameter, y, can be written in terms of its central value $c(y)$ and multiplicative uncertainty factor $u(y)$,

$$y = c(y) \stackrel{\times}{\div} u(x).$$

The relationship between log-normal and normal random variables

$$x = \ln(y)$$

provides the following equation

$$\mu(x) \pm \sigma(x) = \ln(c(y)) \pm \ln(u(y)).$$

Consequently, central value and uncertainty factor of a product of N independent log-normal random variables,

$$y_N = \prod_{i=1}^{N} y_i,$$

are given by

$$c(y_N) = \exp\left(\sum_{i=1}^{N} \ln(c(y_i))\right) \tag{9.5}$$

$$u(y_N) = \exp\left(\sqrt{\sum_{i=1}^{N} \ln^2(u(y_i))}\right). \tag{9.6}$$

It should be pointed out that the central value and the uncertainty factor are not the mean value and the standard deviation of a log-normal random variable. The mean value and standard deviation of a log-normal random variable, y, can be computed from the values of its central value and uncertainty factor,

$$\mu(y) = c(y)\sqrt{\exp(\ln^2(u(y)))} \tag{9.7}$$

$$\sigma(y) = |\mu(y)|\sqrt{\left(\exp(\ln^2(u(y))) - 1\right)}. \tag{9.8}$$

We can also calculate the central value and the uncertainty factor of a log-normal random variables if we have its mean value and standard deviation,

$$c(y) = \mu(y)\sqrt{\frac{\mu^2(y)}{\mu^2(y) + \sigma^2(y)}} \tag{9.9}$$

$$u(y) = \exp\left(\sqrt{\ln\left(\frac{\mu^2(y)+\sigma^2(y)}{\mu^2(y)}\right)}\right). \tag{9.10}$$

Clearly, in the case that uncertainty factor of a parameter equals to one, that parameter is in fact a deterministic parameter.

Now, let us consider again the stochastic model of direct climate forcing. By assuming that each uncertain parameter has log-normal distribution and these parameters are independent each other, we can use equations (9.5 - 9.6) to calculate the central values and uncertainty factors for variables ΔF_R, $\delta_{SO_4^{2-}}$, and $B_{SO_4^{2-}}$ defined by equations (9.2 - 9.4),

$$\begin{aligned} c(B_{SO_4^{2-}}) &= 0.00325417 \\ u(B_{SO_4^{2-}}) &= 1.80435 \\ c(\delta_{SO_4^{2-}}) &= 0.0276605 \\ u(\delta_{SO_4^{2-}}) &= 2.02064 \\ c(\Delta F_R) &= -0.925755 \\ u(\Delta F_R) &= 2.33506, \end{aligned}$$

and using equations (9.7 - 9.8), the corresponding mean values and standard deviations become

$$\begin{aligned} \mu(B_{SO_4^{2-}}) &= 0.0038733 \\ \sigma(B_{SO_4^{2-}}) &= 0.00250033 \\ \mu(\delta_{SO_4^{2-}}) &= 0.0354244 \\ \sigma(\delta_{SO_4^{2-}}) &= 0.028343 \\ \mu(\Delta F_R) &= -1.32636 \\ \sigma(\Delta F_R) &= 1.36087. \end{aligned}$$

We can also use DEMM to solve the this problem. First, we must translate all central values and uncertainty factors for uncertain parameters into polynomial chaos expansions. Since a log-normal random variable y with central value $c(y)$ and uncertainty factor $u(y)$ can be generated from a normal random variable with mean value $\ln(c(y))$ and standard deviation $\ln(u(y))$,

$$y = \exp(\ln(c(y)) + \ln(u(y))\, x),$$

where x is a standard normal random variable, we can compute the skewness, γ_1, and the kurtosis, γ_2, factors for y as follows [112]:

$$\begin{aligned} \gamma_1 &= (\omega-1)^{\frac{1}{2}}(\omega+2) \\ \gamma_2 &= \omega^4 + 2\omega^3 + 2\omega^2 - 3, \end{aligned}$$

where $\omega = \exp(\ln^2(u(y)))$. The mean value and standard deviation of y are obtained from equations (9.7 - 9.8). These values are then used to generate polynomial chaos expansion of y by applying the least-square technique described in chapter 4.

For this problem, we have three stochastic equations with three dependent stochastic

variables: ΔF_R, $\delta_{SO_4^{2-}}$, and $B_{SO_4^{2-}}$, and nine uncertain independent parameters which have log-normal distribution. In this case, we use a four-term polynomial chaos expansion for each uncertain parameter,

$$p_i = p_{i0} + p_{i1}\ \xi_i + p_{i2}\ (\xi_i^2 - 1) + p_{i3}\ (\xi_i^3 - 3\xi_i),$$

where ξ_i is a standard normal random variable, and a thirty-four-term expansion for each dependent stochastic variable,

$$x_i = x_{i0} + \sum_{j=1}^{9} x_{ij}\ \xi_j + x_{i10}\ \xi_2\xi_3 + x_{i11}\ \xi_2\xi_4 + x_{i12}\ \xi_2\xi_6 + x_{i13}\ \xi_3\xi_4 +$$
$$x_{i14}\ \xi_3\xi_6 + x_{i15}\ \xi_4\xi_6 + \sum_{j=16}^{24} x_{ij}\ (\xi_{j-15}^2 - 1) + \sum_{j=25}^{33} x_{ij}\ (\xi_{j-24}^3 - 3\xi_{j-24}).$$

Since the model is multiplicative, we add six product terms of four uncertain parameters with large uncertainty factors to increase the accuracy of representation. DEMM input and generated files for this example are given in appendix F. The results of DEMM are given by

$$\begin{aligned}
\mu(B_{SO_4^{2-}}) &= 0.003873 \\
\sigma(B_{SO_4^{2-}}) &= 0.002460856 \\
\mu(\delta_{SO_4^{2-}}) &= 0.035424 \\
\sigma(\delta_{SO_4^{2-}}) &= 0.02739225 \\
\mu(\Delta F_R) &= -1.326356 \\
\sigma(\Delta F_R) &= 1.236291,
\end{aligned}$$

and using equations (9.9 - 9.10), the corresponding central values and uncertainty factors become

$$\begin{aligned}
c(B_{SO_4^{2-}}) &= 0.00326895 \\
u(B_{SO_4^{2-}}) &= 1.79023 \\
c(\delta_{SO_4^{2-}}) &= 0.0280231 \\
u(\delta_{SO_4^{2-}}) &= 1.98304 \\
c(\Delta F_R) &= -0.970238 \\
u(\Delta F_R) &= 2.20507.
\end{aligned}$$

Comparing these values with the analytical solutions above shows that DEMM can produce quite accurate results for this multiplicative model in which uncertain parameters are represented by their central value and uncertainty factors (see figure 9-3). It should be noted that the resulting 102 deterministic equivalent algebraic equations are solved by the subroutine **broydn** of the Numerical Recipes book [127] with a **DEC Alpha 3000/500** machine in 0.62 second.

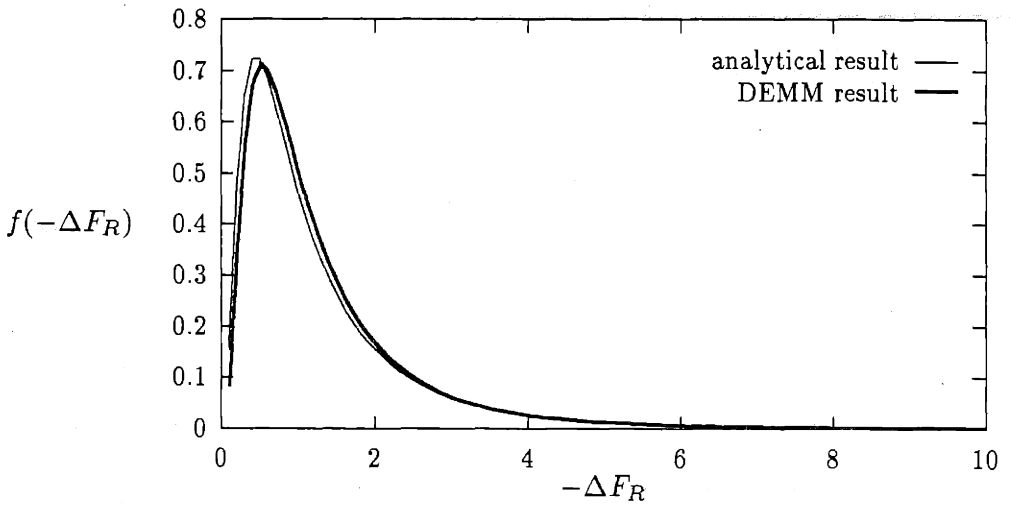


Figure 9-3: Probability density functions of $-\Delta F_R$ derived from analytical method and the DEMM result.

9.1.3 Small gas-phase process flowsheet

A simplified gas-phase process with only slightly nonlinear characteristic given by Näf [113] is considered as an example of the use of DEMM in the environment of an equation-based process simulator. The flowsheet of this process can be seen in figure 9-4. The feed

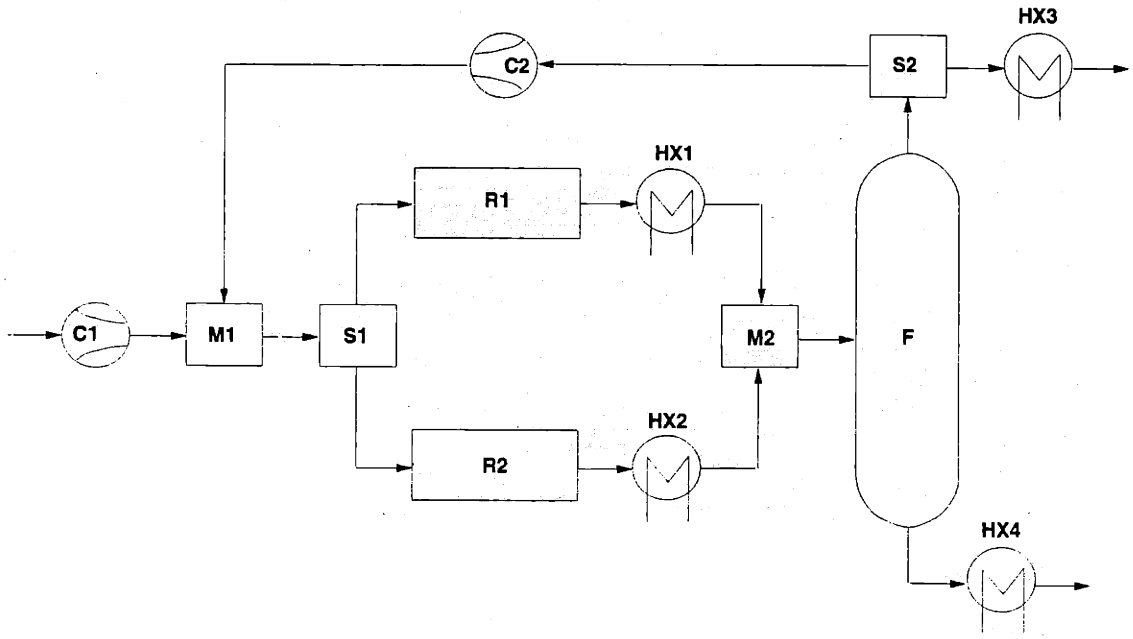


Figure 9-4: Flowsheet of gas-phase process.

stream which contains components A and B is compressed by compressor C1 and mixed with process recycle stream in M1. The outlet of M1 is sent to two parallel reactors R1 and R2 by splitting it with a fixed ratio in S1. A fast and isothermal reaction forming component C takes place in both reactors. The outlets from both reactors are cooled to their boiling point and then condensed in HX1 and HX2. Product C is separated in an isothermal flash drum F. Unreacted component A and B are recycled through the compressor C2, and a fraction of the recycle stream is purged via splitter S2. Equations for each of these units are given below.

- Compressor
  $T_{out} = T_{in} \left(\frac{P_{out}}{P_{in}}\right)^{\frac{1}{2}}$.
- Reactor
  Reaction mechanism: $A + B \rightarrow 2\,C$.
  Conversion for R1: $U = 0.5 + 2.5\ 10^{-4}\ T + 4\ 10^{-3}\ P$.
  Conversion for R2: $U = 0.6 + 2.5\ 10^{-4}\ T + 4\ 10^{-3}\ P$
- Flash drum
  $K_A = \frac{y_A}{x_A} = 16$
  $K_B = \frac{y_B}{x_B} = 8$
  $K_C = \frac{y_C}{x_C} = 0.4$

In this case, let us assume that there are four uncertain parameters which have independent normal distribution: It means that we need to use four standard Gaussian random

Flowrate Feed	=	N($\mu = 100, \sigma = 3.33$) kmol/h.
x_A Feed	=	N($\mu = 0.475, \sigma = 0.00833$).
Cooling water to HX1	=	N($\mu = 298.0, \sigma = 1.667$) K.
Cooling water to HX2	=	N($\mu = 298.0, \sigma = 1.667$) K.

variables to represent the response variables. For this problem, we will use a nine-term polynomial chaos expansion for each response variable in the model,

$$x_i = x_{i0} + \sum_{j=1}^{4} x_{ij}\ \xi_j + \sum_{j=5}^{8} x_{ij}\ (\xi_{j-4}^2 - 1), \tag{9.11}$$

where ξ_1 and ξ_2 represent the uncertainty of temperature of cooling water to HX1 and HX2 respectively, and ξ_3 and ξ_4 denote the uncertainty of component A in the feed and the flowrate of feed. Before we solve the stochastic model, it would be useful to see the result of deterministic model in which the values of uncertain parameters are taken at their mean values.

The deterministic model consists of 183 active equations, and this deterministic flowsheet is solved using **ABACUSS** [10], an equation-based process simulator. The results for seven response variables of interest are given as follows:

Total product flowrate, v_1	=	96.98124 kmol/h
Product purity (x_C), v_2	=	0.93325
Purge flowrate, v_3	=	3.01876 kmol/h
Capacity of reactor R1, v_4	=	64.83083 kmol/h
Capacity of reactor R2, v_5	=	92.52556 kmol/h
Temperature cooling water return HX1, v_6	=	314.98117 K
Temperature cooling water return HX2, v_7	=	314.96214 K.

It took 1.832 seconds to generate these results with a **DECstation 5000/20**.

With nine-term polynomial chaos expansions (DEMM input file for this example is in appendix F), the number of active equations for the corresponding deterministic equivalent model is 2147, and it required 24.344 seconds to solve this model with **DECstation 5000/20**. It should be noted that we may reduce the computational time for solving such a deterministic equivalent model by considering the sparseness structure of it. For those seven variables of interest, the values of nine coefficients of the polynomial chaos expansions are given inside the corresponding brackets below,

$$\begin{aligned}
v_1 &= (96.91948, 0.0, 0.0, 1.38010, 3.22953, 0.0, 0.0, -0.06061, 0.0) \\
v_2 &= (0.93335, 0.0, 0.0, 0.002, 0.0, 0.0, 0.0, 0.0001, 0.0) \\
v_3 &= (3.08052, 0.0, 0.0, -0.54710, 0.10047, 0.0, 0.0, 0.06061, 0.0) \\
v_4 &= (65.31433, 0.0, 0.0, -3.93953, 2.15847, 0.0, 0.0, 0.47446, 0.0) \\
v_5 &= (93.21559, 0.0, 0.0, -5.62244, 3.08053, 0.0, 0.0, 0.67714, 0.0) \\
v_6 &= (315.04147, 1.66700, 0.0, -0.63516, 0.56542, 0.0, 0.0, 0.05917, 0.0)
\end{aligned}$$

$$v_7 = (315.02237, 0.0, 1.66700, -0.63444, 0.56479, 0.0, 0.0, 0.05911, 0.0).$$

These results show that in the presence of such uncertainties the capacities of both reactors should be increased accordingly to obtain those expected values. Furthermore, for this model and such types of uncertainties, only the variability of component A in the feed, ξ_3, effects the product purity, v_2. In order to increase product purity, we need to increase the fraction of component A in the feed.

From chapter 7, we know that in Galerkin's approach, the stochastic variational computation is scaled according to the related probability density function. In other words, the error of approximation has a smaller weight for regions with smaller value of probability density function. Since it is recommended to refine the solution from Galerkin's approach whenever we want to calculate probability values, such as the stochastic flexibility index of a system, we need to apply the collocation refinement technique to this problem. We choose two collocation points $\pm\sqrt{3}$, which are non-zero roots of the third order Hermite polynomial. As an example, we will consider the refinement of the result of capacity of reactor R1, v_4, with respect to the uncertainty of component A in the feed, ξ_3.

In order to apply the collocation refinement technique, we solve two deterministic problems with parameters at their mean value except for parameter x_A in the feed. In the first deterministic problem, parameter x_A takes value of $0.475 + \sqrt{3}\ 0.00833$, and in the second problem, this parameter equals to $0.475 - \sqrt{3}\ 0.00833$. The values of v_4 for these two problems are 54.34098 and 85.44959 respectively. A minimization problem can be devised to calculate the optimal coefficients for ξ_3 and $\xi_3^2 - 1$,

$$\begin{aligned} \min_{a_1,a_2} \quad & f^2 \\ \text{s.t.} \quad & a_0 = 65.31433 \\ & a_1^2 + 2a_2^2 - 15.9701 = f \\ & a_0 + \sqrt{3}a_1 + 2a_2 = 54.34098 \\ & a_0 - \sqrt{3}a_1 + 2a_2 = 85.44959, \end{aligned}$$

where 65.31433 is the mean value of v_4, and 15.9701 is a fraction of variance of v_4 from the ξ_3 terms before the refinement. However, since in this case we have two collocation points, $p_1 = \sqrt{3}$ and $p_2 = -\sqrt{3}$, and two unknown coefficients, we can directly solve two equations,

$$\begin{aligned} a_0 + p_1 a_1 + a_2(p_1^2 - 1) &= r_1 \\ a_0 + p_2 a_1 + a_2(p_2^2 - 1) &= r_2, \end{aligned}$$

for these two coefficients a_1 and a_2,

$$\begin{aligned} a_1 &= \frac{(p_2^2 - 1)(r_1 - a_0) + (p_1^2 - 1)(a_0 - r_2)}{p_1 p_2^2 - p_1^2 p_2 + p_2 - p_1} \\ a_2 &= \frac{p_2(a_0 - r_1) + p_1(r_2 - a_0)}{p_1 p_2^2 - p_1^2 p_2 + p_2 - p_1}, \end{aligned}$$

where for this example $r_1 = 54.34098$ and $r_2 = 85.44959$. The coefficients for v_4 after this refinement become

$$v_4 = (65.31433, 0.0, 0.0, -8.98029, 2.15847, 0.0, 0.0, 2.29048, 0.0).$$

Furthermore, we may refine this v_4 with respect to ξ_4, and the result of refinement is given by

$$v_4 = (65.31433, 0.0, 0.0, -8.98029, 2.15891, 0.0, 0.0, 2.29048, -0.24170).$$

Similarly, we can refine the coefficients of v_2 with respect to ξ_3, and v_1, v_3 and v_5 with respect to ξ_3 and ξ_4, and the coefficients of v_6 with respect to ξ_1, ξ_3, and ξ_4, and the coefficients of v_7 with respect to ξ_2, ξ_3, and ξ_4. The results from these refinements are given as follows:

$$\begin{aligned}
v_1 &= (96.91948, 0.0, 0.0, 1.1472, 3.22954, 0.0, 0.0, -0.29260, 0.03096) \\
v_2 &= (0.93335, 0.0, 0.0, 0.00373, 0.0, 0.0, 0.0, 0.00058, 0.0) \\
v_3 &= (3.08052, 0.0, 0.0, -1.14720, 0.10053, 0.0, 0.0, 0.29260, -0.03088) \\
v_5 &= (93.21559, 0.0, 0.0, -12.8165, 3.08117, 0.0, 0.0, 3.26894, -0.34495) \\
v_6 &= (315.04147, 1.66681, 0.0, -1.12001, 0.56549, -0.03015, 0.0, 0.28567, -0.03014) \\
v_7 &= (315.02237, 0.0, 1.66681, -1.11876, 0.56485, 0.0, -0.03012, 0.28535, -0.0301).
\end{aligned}$$

We can calculate the stochastic flexibility index with the following inequality constraints [113] by using the above expressions for response variables and 2750 Hammersley Wozniakowski sampling points. In this approach, we do not have to solve the flowsheet

Total Product flowrate	$>$	90 kmol/h
Product purity (x_C)	$>$	0.9
Purge flowrate	$<$	5% of product flowrate
Capacity of reactor R1	$<$	70 kmol/h
Capacity of reactor R2	$<$	100 kmol/h
Temperature of cooling water return HX1	$<$	Temperature hot inlet - 10 K
Temperature of cooling water return HX2	$<$	Temperature hot inlet - 10 K

for every sampling point because we only need to evaluate the inequalities based on the chance-constrained formulation (see chapter 6). The results are given by:

$$\begin{aligned}
P(v_1 \leq 90) &= 0.976 \\
P(v_2 \geq 0.9) &= 1.0 \\
P(v_3 \leq 0.05 v_1) &= 0.9015 \\
P(v_4 \leq 70) &= 0.7422 \\
P(v_5 \leq 100) &= 0.7407 \\
P(v_6 \leq T_{hot} - 10) &= 1.0 \\
P(v_7 \leq T_{hot} - 10) &= 1.0,
\end{aligned}$$

and the probability that all these constraints are simultaneously satisfied is 0.7295. If we use 5000 or 10000 sampling points, we will have the stochastic flexibility index for this system to be 0.7310 or 0.7311 respectively. Näf obtained almost the same result which is 0.692 ± 0.017 by solving the flowsheet 2741 times with a shuffling pseudo-random number generator [113].

The use of collocation approach to this type of problem is not only for refining the result from Gallerkin approach, but also can be extended to conducting uncertainty analysis of

implicit or black-box models. An example of implicit or black-box models is a flowsheet model developed in the environment of modular-based process simulators. Since many existing commercial process simulators are of modular-based type, it is of interest to see the application of collocation approach to such a black-box model. To illustrate this idea, we will again consider the small gas-phase flowsheet model, but this time we will treat it as a black-box model in which we do not have access to its model equations.

The nine-term polynomial chaos expansion given by equation (9.11) is still used to represent each dependent variable of interest. From the structure of this representation, we may choose the roots of third order Hermite polynomial as our collocation points. Since we have four uncertain parameters and nine unknown coefficients in this example, we need to use nine collocation points of four dimensional type,

$$\begin{aligned}
p_1 &= (0,0,0,0) \\
p_2 &= (\sqrt{3},0,0,0) \\
p_3 &= (-\sqrt{3},0,0,0) \\
p_4 &= (0,\sqrt{3},0,0) \\
p_5 &= (0,-\sqrt{3},0,0) \\
p_6 &= (0,0,\sqrt{3},0) \\
p_7 &= (0,0,-\sqrt{3},0) \\
p_8 &= (0,0,0,\sqrt{3}) \\
p_9 &= (0,0,0,-\sqrt{3}).
\end{aligned}$$

We then solve the deterministic flowsheet model nine times. The value of the j-th uncertain parameter in the i-th simulation is given by

$$c_{i,j} = \mu(u_j) + \sigma(u_j) p_{i,j},$$

where u_j is the j-th uncertain parameter. From these nine simulations, we clearly have nine values for each dependent variable of interest, $v_{k,i}, i = 1, \ldots, 9$. To calculate the values of nine coefficients for the k-th dependent variable, we need to solve the following set of linear equations:

$$a_{k,0} + \sum_{j=1}^{4} a_{k,1,j}\ p_{i,j} + \sum_{j=1}^{4} a_{k,2,j}\ (p_{i,j}^2 - 1) = v_{k,i};\quad i = 1, \ldots, 9.$$

From these nine simulations, the values of dependent variables of interest are obtained as follows

$$\begin{aligned}
v_1 &= (96.98124, 96.98124, 96.98124, 96.98124, 96.98124, \\
&\quad 98.32128, 94.34727, 102.57512, 91.38766) \\
v_2 &= (0.93325, 0.93325, 0.93325, 0.93325, 0.93325, \\
&\quad 0.94097, 0.92805, 0.93325, 0.93325) \\
v_3 &= (3.01876, 3.01876, 3.01876, 3.01876, 3.01876, \\
&\quad 1.67872, 5.65273, 3.19288, 2.84464) \\
v_4 &= (64.83083, 64.83083, 64.83083, 64.83083, 64.83083,
\end{aligned}$$

$$
\begin{aligned}
& 54.34098, 85.44959, 68.57027, 61.09158) \\
v_5 &= (92.52556, 92.52556, 92.52556, 92.52556, 92.52556, \\
& 77.55460, 121.95233, 97.86243, 87.18896) \\
v_6 &= (314.98117, 317.86817, 312.09417, 314.98117, 314.98117, \\
& 313.67288, 317.55272, 315.96064, 314.00174) \\
v_7 &= (314.96214, 314.96214, 314.96214, 317.84914, 312.07514, \\
& 313.65532, 317.53082, 315.94052, 313.98382) \\
v_8 &= (557.51975, 557.51975, 557.51975, 557.51975, 557.51975, \\
& 584.06622, 524.33659, 557.51975, 557.51975) \\
v_9 &= (557.51975, 557.51975, 557.51975, 557.51975, 557.51975, \\
& 584.06622, 524.33659, 557.51975, 557.51975),
\end{aligned}
$$

where v_1 to v_7 are the same as given previously, and v_8 and v_9 are temperatures of hot stream to HX1 and HX2 respectively. Therefore, the values of nine coefficients of each dependent variable are computed from solving nine equations above

$$
\begin{aligned}
v_1 &= (96.7656, 0.0, 0.0, 1.1472, 3.22954, 0.0, 0.0, -0.215655, 0.00005) \\
v_2 &= (0.93367, 0.0, 0.0, 0.00372968, 0.0, 0.0, 0.0, 0.00042, 0.0) \\
v_3 &= (3.23441, 0.0, 0.0, -1.1472, 0.100528, 0.0, 0.0, 0.215655, 0.0) \\
v_4 &= (66.519, 0.0, 0.0, -8.98028, 2.15891, 0.0, 0.0, 1.68815, 0.0000316667) \\
v_5 &= (94.9349, 0.0, 0.0, -12.8165, 3.08117, 0.0, 0.0, 2.4093, 0.000045) \\
v_6 &= (315.192, 1.66681, 0.0, -1.12001, 0.565486, 0.0, 0.0, 0.210543, 0.0) \\
v_7 &= (315.172, 0.0, 1.66681, -1.11876, 0.564851, 0.0, 0.0, 0.21031, 0.00001) \\
v_8 &= (556.414, 0.0, 0.0, 17.2425, 0.0, 0.0, 0.0, -1.10612, 0.0) \\
v_9 &= (556.414, 0.0, 0.0, 17.2425, 0.0, 0.0, 0.0, -1.10612, 0.0).
\end{aligned}
$$

Similar to the result from Galerkin's approach, these expressions for response variables can be used to compute the stochastic flexibility index of the system. We again apply the Hammersley Wozniakowski sampling points to the chance-constrained formulation, and using 2750 points we have

$$
\begin{aligned}
P(v_1 \leq 90) &= 0.9731 \\
P(v_2 \geq 0.9) &= 1.0 \\
P(v_3 \leq 0.05 v_1) &= 0.8924 \\
P(v_4 \leq 70) &= 0.6964 \\
P(v_5 \leq 100) &= 0.6931 \\
P(v_6 \leq v_8 - 10) &= 1.0 \\
P(v_7 \leq v_9 - 10) &= 1.0.
\end{aligned}
$$

The stochastic flexibility index becomes 0.680, and for the case of 5000 and 10000 sampling points we obtain 0.6814 and 0.6813 respectively. This result suggests that by using a good representation for dependent variables as well as solving the black-box model at collocation points we can produce a quite accurate solution. It also means that the application of

collocation approach, or in general the response surface methodology, to conduct uncertainty analysis of implicit or black-box models looks promising. This application clearly becomes more important when the evaluation of such implicit models is expensive.

Figures 9-5 to 9-8 show the simulation of results that obtained from the collocation method. They are generated in the S language environment, and the commands are as follows:

```
> x1 <- rnorm(2741)
> x2 <- rnorm(2741)
> x3 <- rnorm(2741)
> x4 <- rnorm(2741)
> mfa <- 0.475 + 0.00833*x3
> pfr <- 100.0 + 3.33*x4
> v4 <- 66.519 - 8.98028*x3 + 2.15891*x4 + 1.68815*(x3*x3 - 1) +
+ 0.0000316667*(x4*x4-1)
> v2 <- 0.93367 + 0.00372968*x3 + 0.00042*(x3*x3-1)
> plot(pfr,v4,xlab="Plant Feed Flowrate (kmol/h)",
+ ylab="Flowrate R1 (kmol/h)")
> plot(mfa,v4,xlab="Mole Fraction of A in Feed",
+ ylab="Flowrate R1 (kmol/h)")
> plot(mfa,v2,xlab="Mole Fraction of A in Feed",
+ ylab="Product Purity (Mole Fraction C)")
> plot(density(v4,n=200),type="l",xlab="Flowrate (kmol/h)",
+ ylab="Probability density function of Flowrate to R1")
```

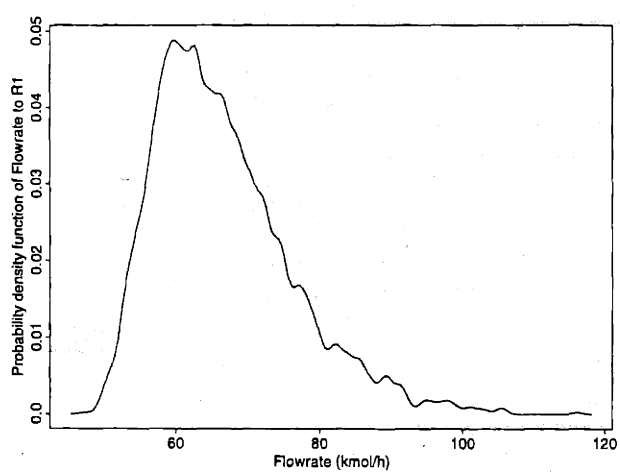


Figure 9-5: The probability density function of flow rate to reactor R1.

These figures suggest that the results by solving the deterministic process at 9 collocation points are comparable to those by solving the same problem at 2741 sampling points [113].

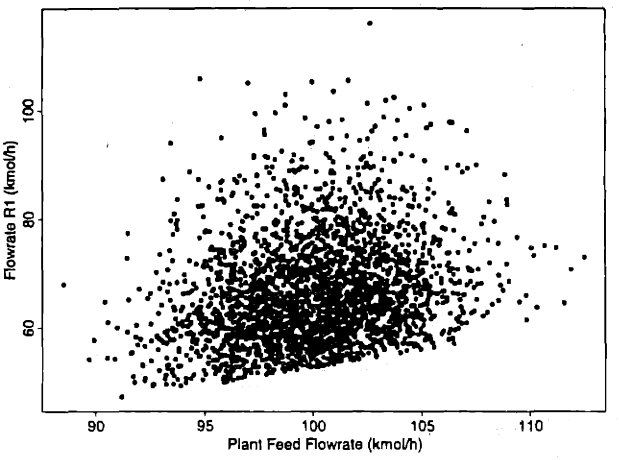


Figure 9-6: Scatterplot of flow rate of reactor R1 as a function of plant flow rate.

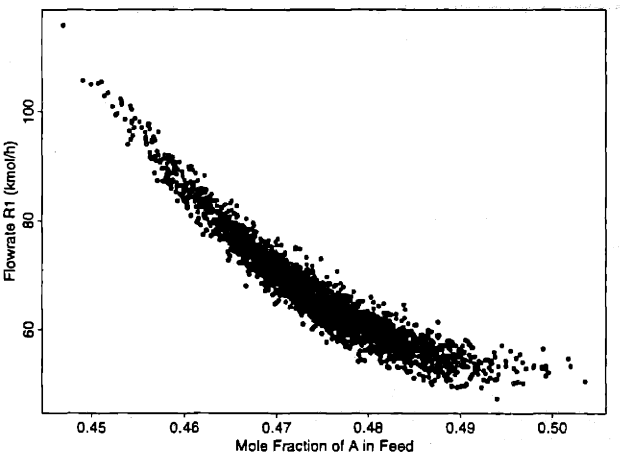


Figure 9-7: Scatterplot of flow rate of reactor R1 as a function of mole fraction of A in feed.

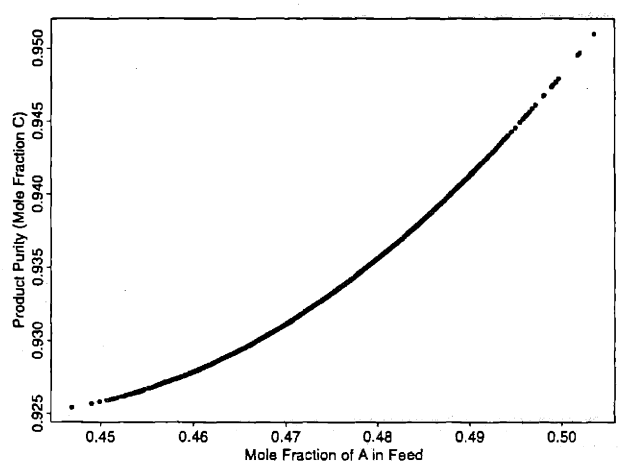


Figure 9-8: Scatterplot of product purity as a function of mole fraction of A in feed.

9.1.4 Multiple stationary points example

An open three-variables mass action kinetics taken from Willamowski and Rössler [191] is considered below.

$$\begin{aligned} \dot{x} &= x\,(a_1 - k_1\,x - z - y) + k_2\,y^2 + a_3 \\ \dot{y} &= y\,(x - k_2\,y - a_5) + a_2 \\ \dot{z} &= z\,(a_4 - x - k_5\,z) + a_3 \end{aligned}$$

The deterministic model with specified values of parameters has three fixed points: (0.0033, 0.0010, 32.9939), (1.7120, 0.0012, 29.5766), and (11.0456, 18.0372, 10.9106). The first and third fixed points are stable, and the second fixed point is a saddle point. Stability of the third fixed point can be altered by changing the value of parameters. For example, as the value of parameter k_2 decreases, the third fixed point becomes a saddle point. Therefore, the number of reachable stationary points may change from two to one for different values of k_2.

Figure 9-9 shows the numerical simulation of this system of differential equations using **lsode** [67] integration routine (see appendix F for DEMM input file of this example). The second stationary point is reachable. However, when the value of k_2 is uncertain, the second stationary point (the third fixed point) is unreachable (see figure 9-10). The expected trajectory is drawn based on the assumption that k_2 has mean value of 0.058, and its second, third, and fourth central moments equal to 2.634×10^{-5}, 1.3936×10^{-7}, and 3.428×10^{-9} respectively. From these moments, the first four coefficients of polynomial chaos

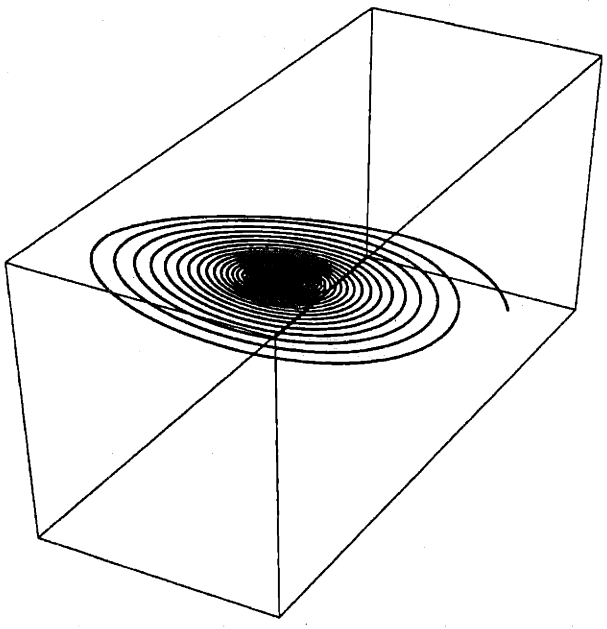


Figure 9-9: Deterministic trajectory near the third fixed point. Parameters: $k_1 = 0.25$, $k_2 = 0.058$, $k_5 = 0.5$, $a_1 = 30$, $a_2 = a_3 = 0.01$, $a_4 = 16.5$, $a5 = 10$. Initial conditions: $x(0) = 12$, $y(0) = 22$, $z(0) = 8$, $t_{end} = 30$. Numerical simulation on a DEC 3000 model 500 computer.

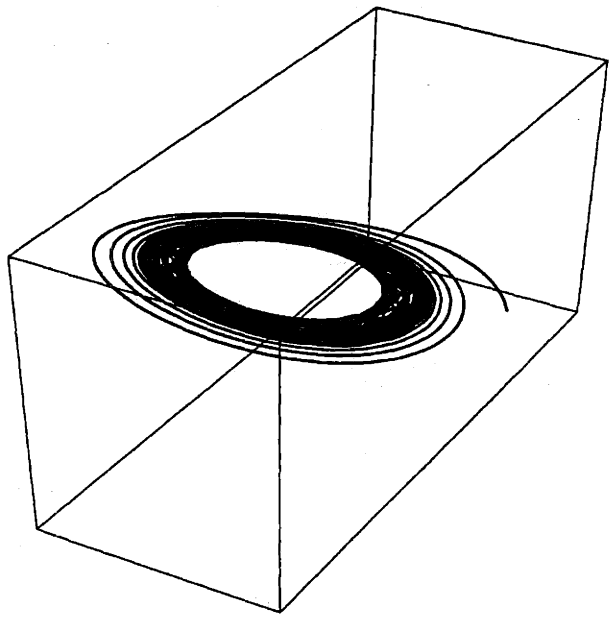


Figure 9-10: Expected trajectory near the third fixed point when k_2 value is uncertain. Parameters: $k_{20} = 0.058$, $k_{21} = 0.005$, $k_{22} = 0.0008$, $k_{23} = 0.0001$.

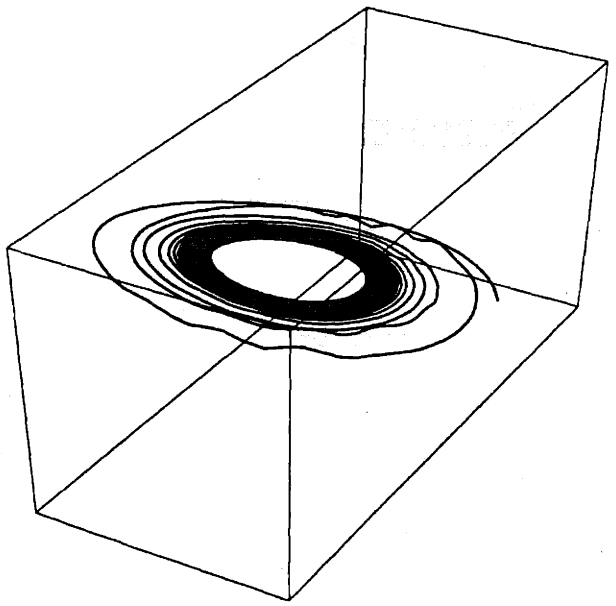


Figure 9-11: Expected trajectory near the third fixed point when both k_2 and $y(0)$ values are uncertain. Initial conditions: $y_0(0) = 22$, $y_2(0) = 2$, $y_1(0) = y_3(0) = y_4(0) = y_5(0) = y_6(0) = y_7(0) = 0$.

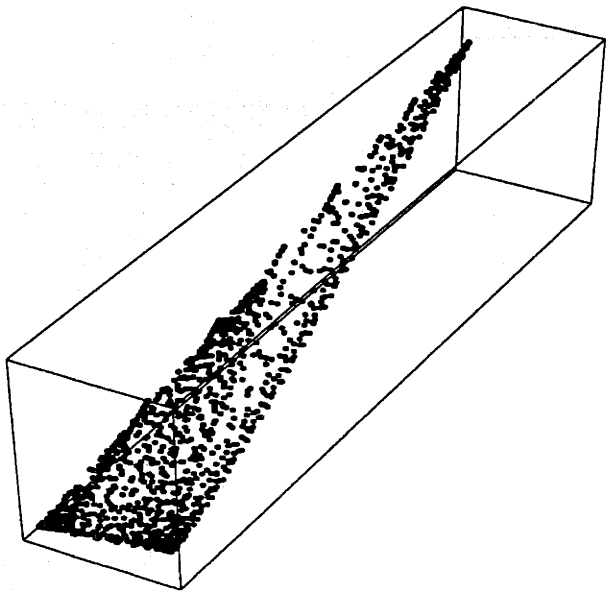


Figure 9-12: Variance trajectory near the third fixed point for case variance of k_2 is small. Parameters: $k_{20} = 0.058$, $k_{21} = 10^{-7}$, $k_{22} = 10^{-8}$, $k_{23} = 10^{-10}$.

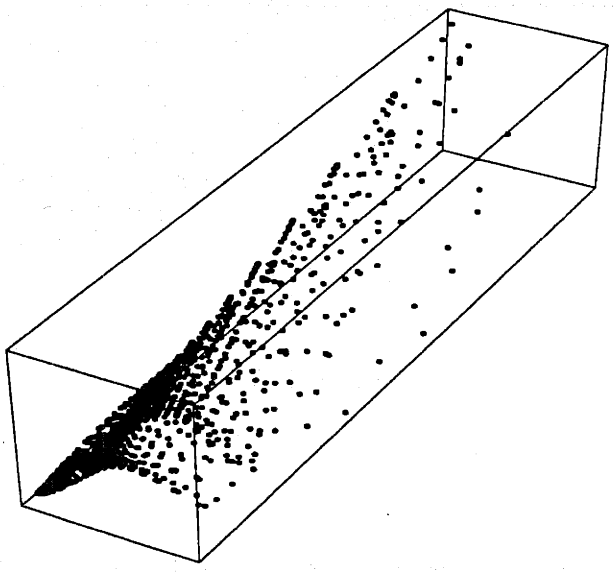


Figure 9-13: Variance trajectory near the third fixed point for case both variances of k_2 and $y(0)$ are small. Initial conditions: $y_0(0) = 22$, $y_2(0) = 10^{-4}$, $y_1(0) = y_3(0) = y_4(0) = y_5(0) = y_6(0) = y_7(0) = 0$.

expansion for parameter k_2 can be calculated by solving four moment equations:

$$\begin{aligned} E(k_2) &= 0.058 \\ E((k_2 - E(k_2))^2) &= 2.634 \times 10^{-5} \\ E((k_2 - E(k_2))^3) &= 1.3936 \times 10^{-7} \\ E((k_2 - E(k_2))^4) &= 3.428 \times 10^{-9} \end{aligned}$$

These coefficients are: $k_{20} = 0.058$, $k_{21} = 0.005$, $k_{22} = 0.0008$, and $k_{23} = 0.0001$.

Uncertainty could also appear in the value of initial conditions. When $y(0)$ is a Gaussian distributed random variable (independent to k_2) with standard deviation 2.0, the third fixed point is still unreachable. Figure 9-11 shows the expected trajectory for this system. The difference between figure 9-10 and figure 9-11 is the appearance of small waves at the beginning of trajectory in figure 9-11. It should be noted that both third expected fixed points in figure 9-10 and figure 9-11 have exactly the same position, these are at (11.0810, 18.6366, 10.4409). Even though the position of this third expected fixed point and the third fixed point from deterministic analysis are only a little bit different, the different reachable conditions of these fixed points are important to be considered.

Another important point for this example is that uncertainty in the initial conditions affects the behavior of variance trajectory. Such an uncertainty can disturb the structure of the trajectory. Figure 9-12 and figure 9-13 show these disturbances. It seems that at the time when the variance trajectory of a system with parametric uncertainty is still in good structure, the variance trajectory of a system with parametric and input uncertainty has already lost its structure.

This simple example of mass action kinetics system suggests that deterministic analysis is not enough to have a reasonable prediction of the behavior of non-linear systems with uncertain parameters and initial conditions, especially when the value of parameters is near the bifurcation value. On the other hand, stochastic analyses can provide more information, such as the moments, or the probability of a trajectory. These kinds of information along with a good assumption on the distribution of uncertain parameters or inputs will be useful to reduce the bias of modeling processes.

9.1.5 Generalized reaction mechanism for photochemical smog

Photochemical smog is the term given to the mixture of reactants and products in the atmosphere that results from the interaction of organics with oxides of nitrogen. A generalized reaction mechanism for photochemical smog is outlined in table 9.3 [140]. To generate the temporal concentration dynamics of RH, $RCHO$, NO, NO_2, and O_3, the following set of initial concentrations is considered,

$$\begin{aligned} [RH]_0 &= 2.0 \text{ ppm} \\ [RCHO]_0 &= 2.0 \text{ ppm} \\ [NO]_0 &= 0.5 \text{ ppm} \\ [NO_2]_0 &= 0.1. \text{ ppm} \end{aligned}$$

First, let use see the result from treating the reaction rate constants as deterministic

Reaction	Rate constant (ppm-min units)(298 K)
$NO_2 + h\nu \rightarrow NO + O$	0.555 min^{-1}
$O + O_2 + M \rightarrow O_3 + M$	2.183×10^{-5}
$NO + O_3 \rightarrow NO_2 + O_2$	26.59
$RH + OH\cdot \rightarrow RO_2\cdot + H_2O$	3.775×10^3
$RCHO + OH\cdot \rightarrow RC(O)O_2\cdot + H_2O$	2.341×10^4
$RCHO + h\nu \rightarrow RO_2\cdot + HO_2\cdot + CO$	$1.91 \times 10^{-4} \text{ min}^{-1}$
$HO_2\cdot + NO \rightarrow NO_2 + OH\cdot$	1.214×10^4
$RO_2\cdot + NO \rightarrow NO_2 + RCHO + HO_2\cdot$	1.127×10^4
$RC(O)O_2\cdot + NO \rightarrow NO_2 + RO_2\cdot + CO_2$	1.127×10^4
$OH\cdot + NO_2 \rightarrow HNO_3$	1.613×10^4
$RC(O)O_2\cdot + NO_2 \rightarrow RC(O)O_2NO_2$	6.893×10^3
$RC(O)O_2NO_2 \rightarrow RC(O)O_2\cdot + NO_2$	$2.143 \times 10^{-2} \text{ min}^{-1}$

Table 9.3: Reaction mechanism for photochemical smog.

variables. Figure 9-14 shows the deterministic result of predicted concentrations by solving a generalized reaction mechanism. The maximum of concentration of NO_2 for this case is reached at about 130 min, and it decays due to formation of nitric acid and PAN. Using a **DEC Alpha 3000/500** workstation, it took 0.4 second to simulate this deterministic model with the **lsode** [67] subroutine from 0 to 600 minutes.

In practice, the values of reaction rate constants may contain uncertainty which is inherited from their measurements and models. Furthermore, the values of reaction rate constants for the first and the sixth reactions are, in fact, assumed [140]. Therefore, in the following, we will consider the same model of mechanism but with 20% standard deviation for each value of reaction rate constant. For this problem, we have twelve dependent variables, and also twelve uncertain parameters. Those twelve uncertain parameters are assumed to have Gaussian distributions (see chapter 3 for techniques to obtain distribution functions of reaction rate constants from the measurement of yields), and they are independent of each other. Each dependent variable is approximated by a polynomial chaos

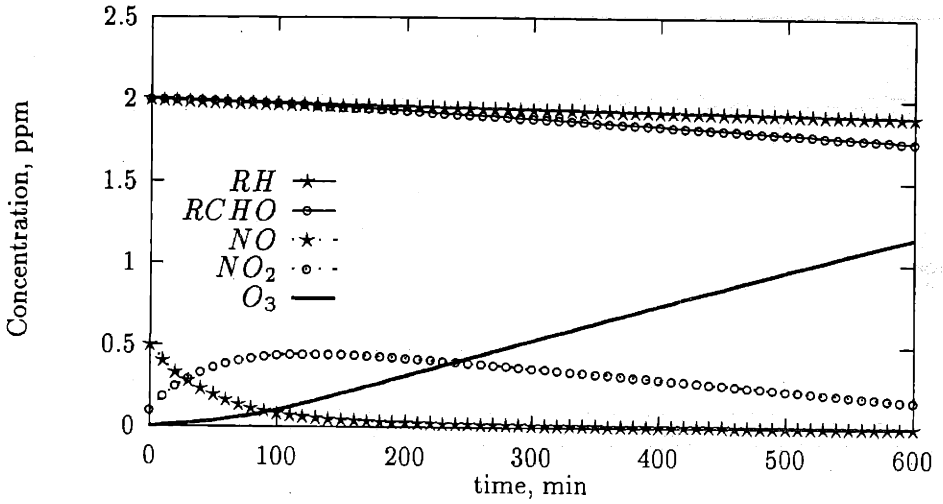


Figure 9-14: Concentrations predicted by a generalized reaction mechanism for photochemical smog. Reaction rate constants are assumed to be deterministic.

expansion with 25 terms,

$$y_i = y_{i,1} + \sum_{j=1}^{12} y_{i,j+1}\ \xi_j + \sum_{j=1}^{12} y_{i,j+13}\ (\xi_j^2 - 1);\quad i = 1, \ldots, 12,$$

where y_i is the i-th dependent variable, and ξ_j represents the j-th standard independent Gaussian random variable for the polynomial chaos expansion. This means that the deterministic equivalent model for this stochastic model consists of 300 ordinary differential equations (DEMM input file for this stochastic model is given in the appendix F). Using the same computer as in the deterministic case, it took 228.5 seconds to simulate these 300 deterministic equivalent ordinary equations with **lsode** from 0 to 600 minutes. Figure 9-15 and 9-16 show the mean values and standard deviations of concentrations of RH, $RCHO$, NO, NO_2, and O_3 respectively. We see that the mean values of concentrations are about the same as the concentrations of the deterministic case. However, the possible ranges of concentrations that are shown by the values of standard deviations of concentrations are quite large for the case of O_3 and NO_2. To assess which uncertain parameters contribute the most to these ranges of uncertainty, we can also use the results of DEMM. As an example, figures 9-17, 9-18, and 9-19 show the first order coefficients of the polynomial chaos expansion of O_3 as functions of time. Those figures suggest that during the time of simulation, the concentration of O_3 is actually very sensitive to the assumed reaction rate constant of the sixth reaction. As a result, we need to put more effort to reduce the uncertainty in that reaction rate constant in order to reduce a large part of the uncertainty in the prediction of concentration of O_3. We also see that the sensitivity index can change its sign during the time of simulation. As an example, increasing the values of the second and the fifth reaction rate constants will raise the concentration of O_3 before the time 300, but will decrease the concentration of O_3 after it.

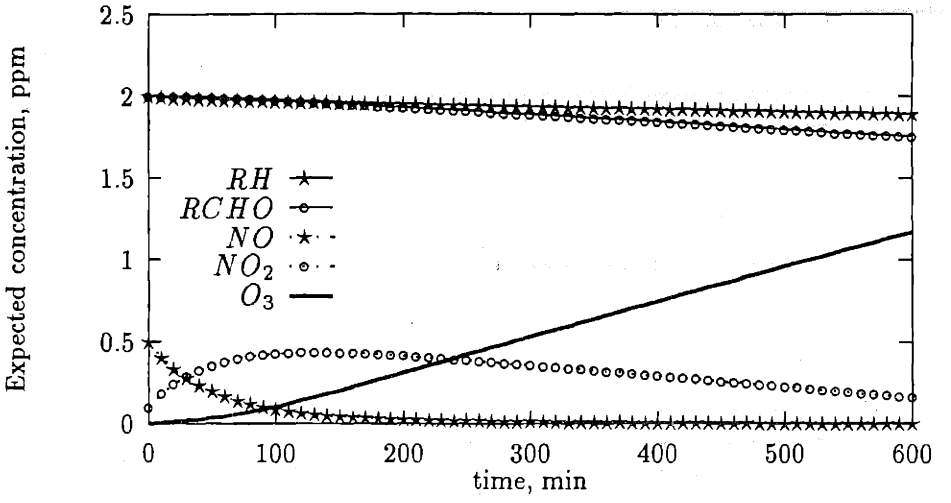


Figure 9-15: Mean values of concentrations predicted by a generalized reaction mechanism for photochemical smog. Reaction rate constants are assumed to be Gaussians with 20% standard deviations.

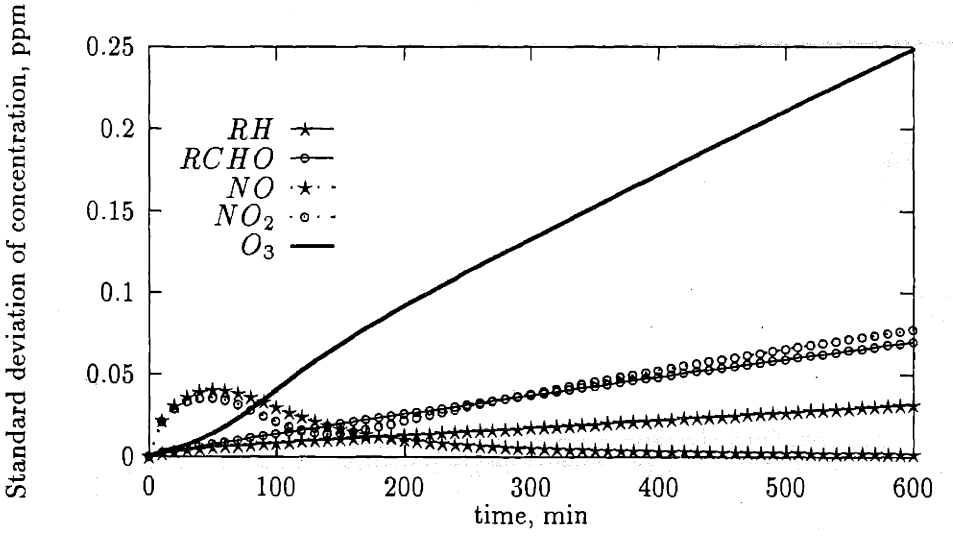


Figure 9-16: Standard deviations of concentrations predicted by a generalized reaction mechanism for photochemical smog. Reaction rate constants are assumed to be Gaussians with 20% standard deviations.

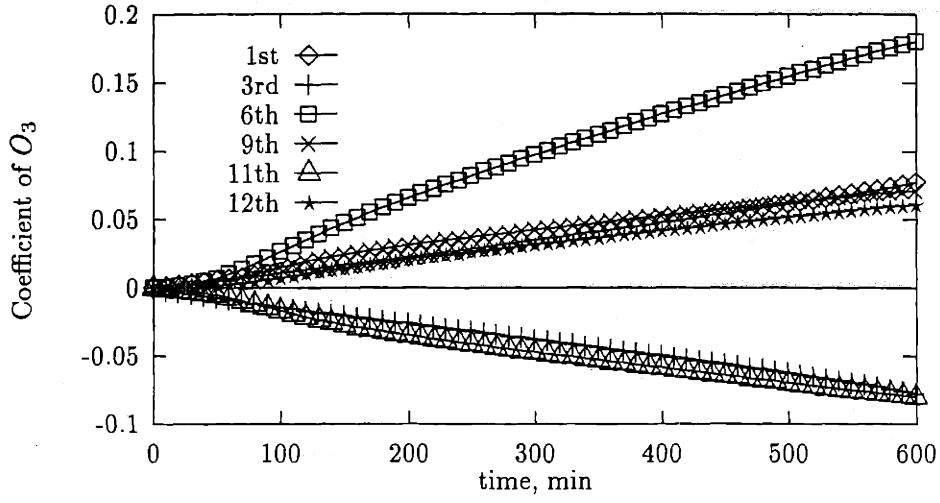


Figure 9-17: Coefficients in the polynomial chaos expansion of O_3 as functions of time. The i-th coefficient represents the contribution of the i-th reaction rate constant.

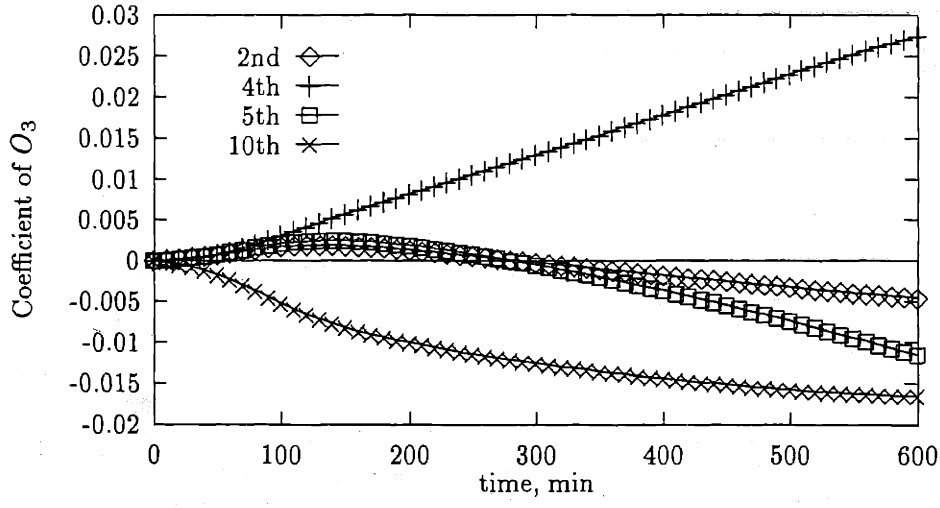


Figure 9-18: Coefficients in the polynomial chaos expansion of O_3 as functions of time. The i-th coefficient represents the contribution of the i-th reaction rate constant.

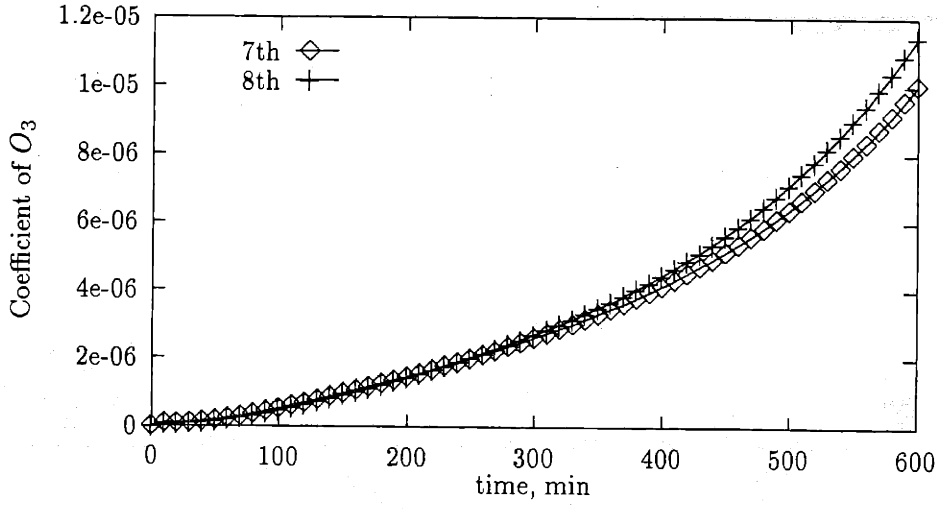


Figure 9-19: Coefficients in the polynomial chaos expansion of O_3 as functions of time. The i-th coefficient represents the contribution of the i-th reaction rate constant.

9.1.6 Reduced H_2/O_2 mechanism

A typical reduced mechanism of a reaction system is considered here. The H_2/O_2 reduced mechanism involves 10 reactions and 7 species. In this case, the initial conditions for some species concentrations are given as follows:

$$\begin{aligned} [O_2]_0 &= 1.04 \times 10^{-6} \text{ mol/cm}^3 \\ [H_2]_0 &= 2.06 \times 10^{-6} \text{ mol/cm}^3 \\ [H_2O]_0 &= 4.285 \times 10^{-3} \text{ mol/cm}^3 \end{aligned}$$

Other initial species concentrations are assumed to be zero. At temperature 500°C and pressure 246 bar, and with the NIST kinetics and current thermochemistry, the values of the reaction rate constants and their uncertainty factors are given in table 9.4.

Reaction	Rate constant (cm^3, mol, s)	
	Mean value	Standard deviation
$OH + H_2 \rightarrow H_2O + H$	6.7628×10^{11}	4.2572×10^{10}
$H_2O \rightarrow OH + H$	2.2426×10^{-12}	3.5433×10^{-13}
$H + O_2 \rightarrow HO_2$	1.0723×10^{13}	1.6942×10^{12}
$OH + HO_2 \rightarrow H_2O + O_2$	2.6608×10^{13}	4.2041×10^{12}
$H_2O + O_2 \rightarrow OH + HO_2$	4.8641×10^{-5}	0
$H_2O_2 + OH \rightarrow H_2O + HO_2$	2.0196×10^{12}	1.5955×10^{11}
$H_2O + HO_2 \rightarrow H_2O_2 + OH$	1.2294×10^{4}	0
$HO_2 + HO_2 \rightarrow H_2O_2 + O_2$	6.7957×10^{11}	4.7910×10^{10}
$H_2O_2 \rightarrow OH + OH$	4.0515×10^{1}	6.4014
$HO_2 + H_2 \rightarrow H_2O_2 + H$	7.3644×10^{6}	7.3644×10^{5}

Table 9.4: Reaction mechanism for H_2/O_2.

In this example, those ten uncertain reaction rate constants are assumed to have Gaussian distributions. Therefore, if we use the first and second order of polynomial chaos expansion for each uncertain reaction rate constant, the approximation of species concentration can be written as follows:

$$y_i = y_{i,1} + \sum_{j=1}^{10} y_{i,j+1}\ \xi_j + \sum_{j=1}^{10} y_{i,j+11}\ (\xi_j^2 - 1);\ \ i = 1, \ldots, 7, \tag{9.12}$$

where y_i is the i-th dependent variable, and ξ_j represents the j-th standard independent Gaussian random variable for the polynomial chaos expansion. It means that we will have $21 \times 7 = 147$ differential equations. The time required to simulate the reactions from time 0 to 10 seconds was 13.52 seconds on a **DEC Alpha 3000/500** workstation with the **lsode** subroutine [67]. It is of interest to note that the deterministic case which contains 7 differential equations can be solved in 0.1 second.

Similar to the example of photochemical smog model, the mean values of species concentrations in this case are about the same as the deterministic species concentrations prediction. Figures 9-20 and 9-21 show both the mean values of H_2 and O_2 concentrations as functions of time and their standard deviations. We see that the value of standard

deviation for H_2 decreases to zero as its mean value goes to zero.

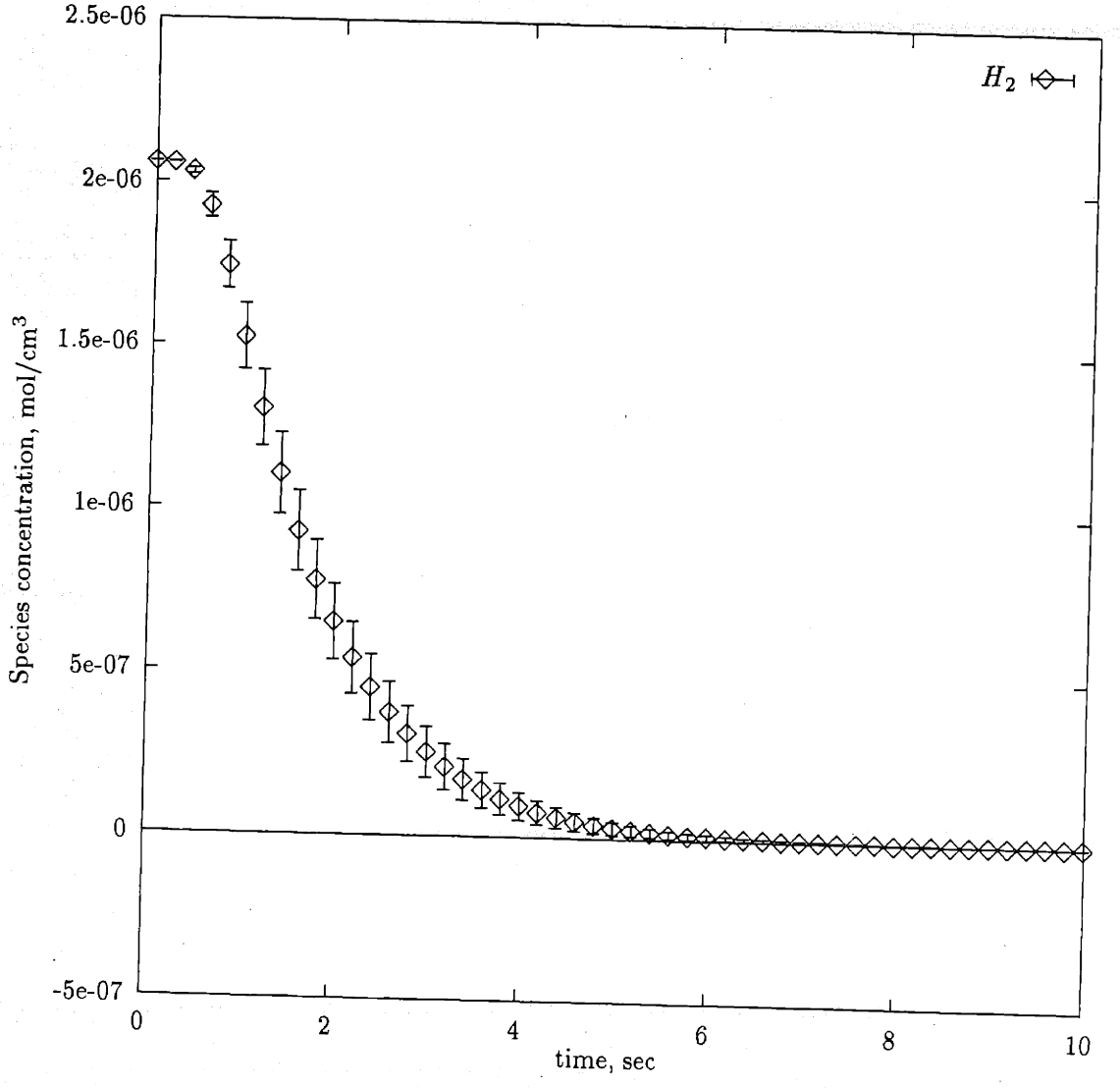


Figure 9-20: H_2/O_2 reduced mechanism predictions; T = 500°C, P = 246 bar; NIST kinetics, current thermochemistry; Gaussian uncertain parameters.

In this example, the first order coefficients of polynomial chaos expansions of species concentrations actually can provide an approximation to the product between first order sensitivity index and the value of standard deviation of the corresponding uncertain reaction rate constant. This product measure may give more insight than the sensitivity index alone because it describes the significance of uncertainty in a reaction rate constant to the prediction of the outputs. Figures 9-22 and 9-23 show such a measure in the case of H_2, and figures 9-24 and 9-25 illustrate the same measure for the case of O_2.

From the plots of that product measure, we see that the uncertainty in the prediction of H_2 and O_2 can be reduced in large by further reducing the uncertainty in the first, the

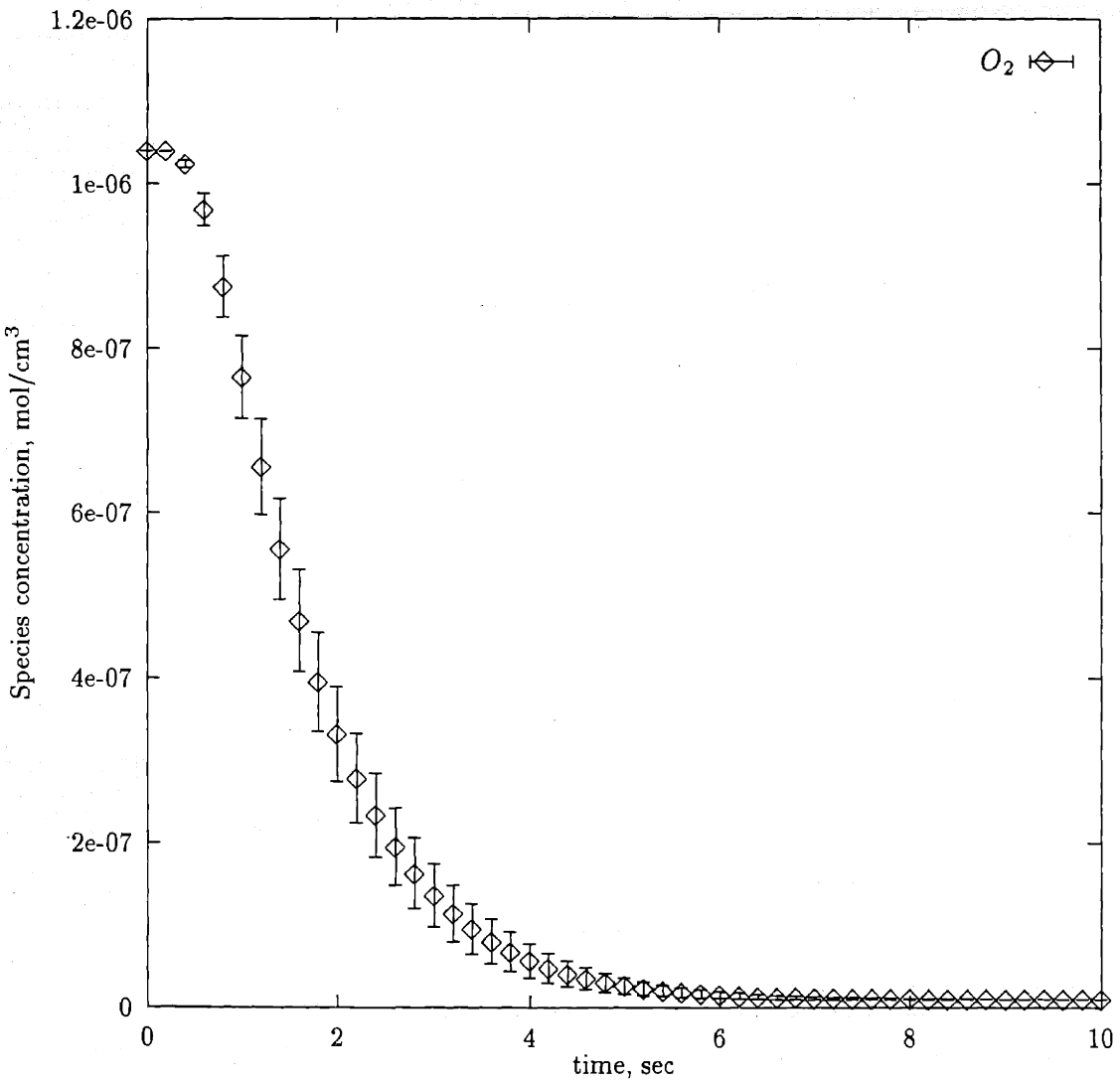


Figure 9-21: H_2/O_2 reduced mechanism predictions; T = 500°C, P = 246 bar; NIST kinetics, current thermochemistry; Gaussian uncertain parameters.

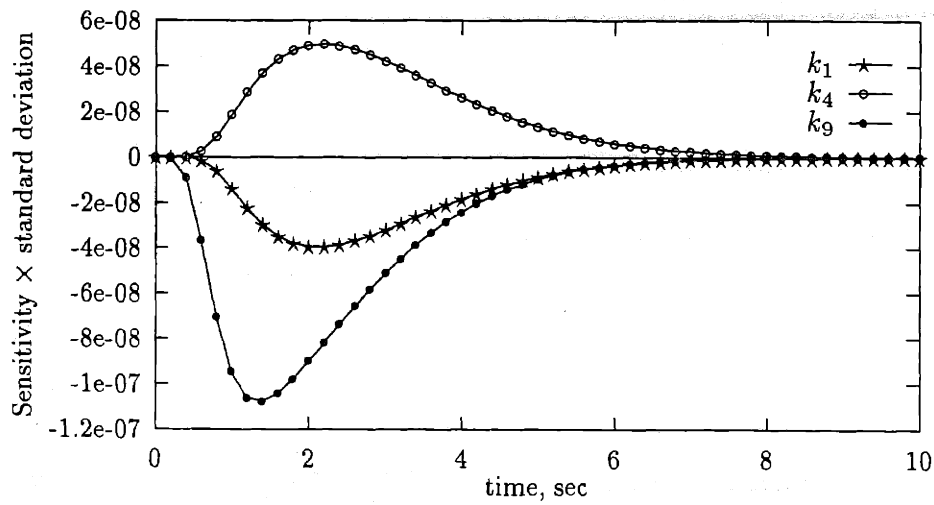


Figure 9-22: H_2/O_2 reduced mechanism predictions; T = 500°C, P = 246 bar; NIST kinetics, current thermochemistry; Gaussian uncertain parameters, H_2 coefficients of uncertainty

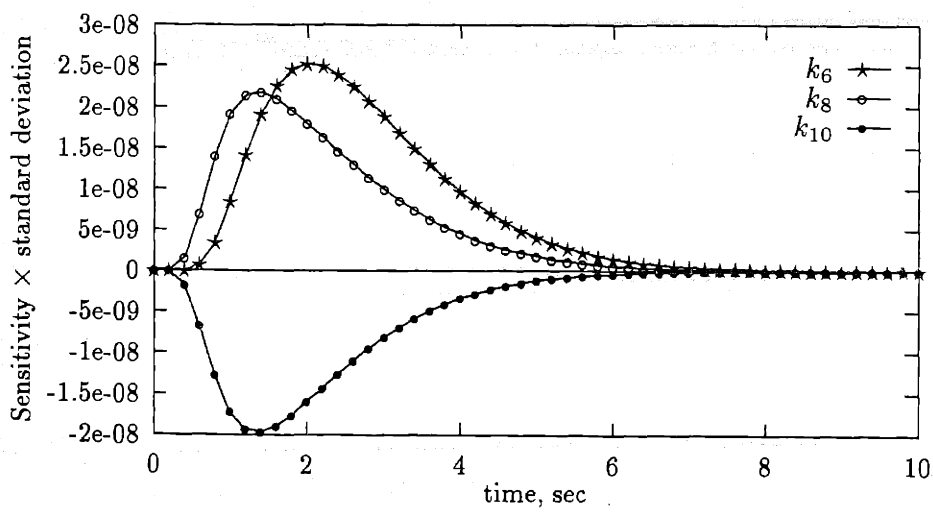


Figure 9-23: H_2/O_2 reduced mechanism predictions; T = 500°C, P = 246 bar; NIST kinetics, current thermochemistry; Gaussian uncertain parameters, H_2 coefficients of uncertainty

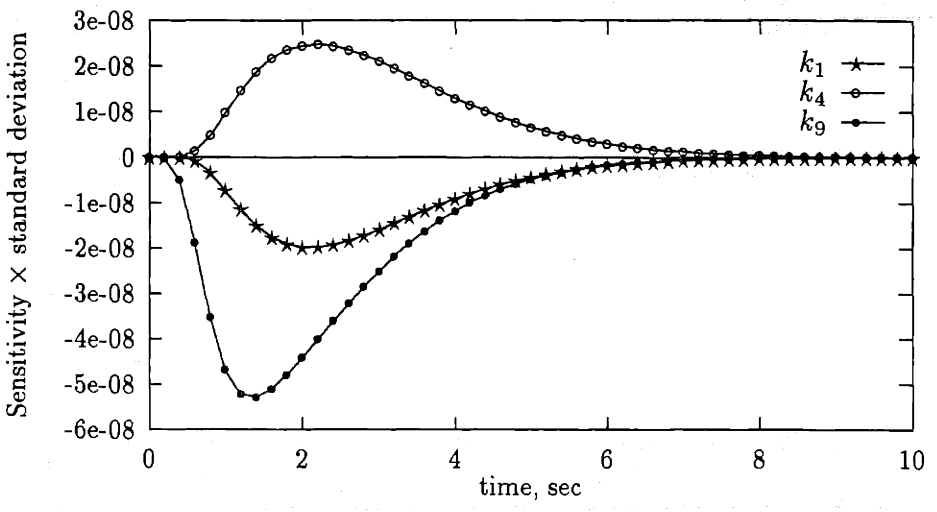


Figure 9-24: H_2/O_2 reduced mechanism predictions; T = 500°C, P = 246 bar; NIST kinetics, current thermochemistry; Gaussian uncertain parameters, O_2 coefficients of uncertainty

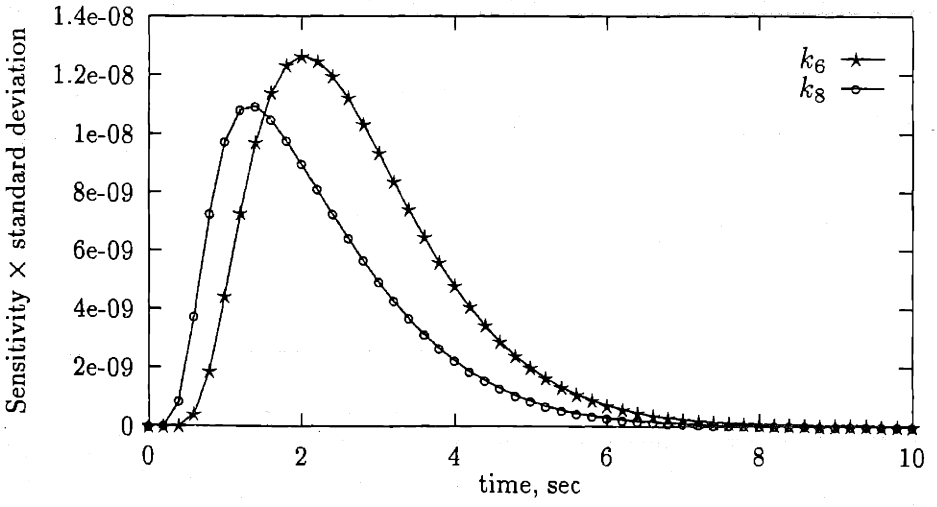


Figure 9-25: H_2/O_2 reduced mechanism predictions; T = 500°C, P = 246 bar; NIST kinetics, current thermochemistry; Gaussian uncertain parameters, O_2 coefficients of uncertainty

fourth, and the ninth reaction rate constants.

9.2 Inequality model examples

9.2.1 Linear inequality model

A linear example is taken from a paper of Straub and Grossmann [163],

$$\begin{aligned} h_1 = &\quad -0.3\theta_1 + 0.2\theta_2 - 5.5 &\leq 0 \\ h_2 = &\quad 0.0075\theta_1 + 0.0375\theta_2 - 4.5 &\leq 0 \\ h_3 = &\quad -0.05\theta_1 - 0.2\theta_2 + 1.0 &\leq 0 \end{aligned}$$

First, we need to expand the uncertain parameters θ_1 and θ_2 into their corresponding polynomial chaos expansions. In this example, the uncertain parameters, θ_1, θ_2, are independent Gaussian distributed random parameters with mean value of 20 and standard deviation of 10, therefore, their polynomial chaos expansion are written as follows,

$$\theta_1 = 20.0 + 10.0\xi_1 \tag{9.13}$$

$$\theta_2 = 20.0 + 10.0\xi_2 \tag{9.14}$$

Hence, the inequality model becomes,

$$h_1 = -7.5 - 3.0\xi_1 + 2.0\xi_2 \leq 0 \tag{9.15}$$

$$h_2 = -2.25 + 0.75\xi_1 + 0.375\xi_2 \leq 0 \tag{9.16}$$

$$h_3 = -4.0 - 0.5\xi_1 - 2.0\xi_2 \leq 0 \tag{9.17}$$

Before we use the Edgeworth series expansion to approximate the distribution functions of slack variables, we may first normalize (mean value = 0, variance = 1) those slack variables, $h_1(\omega), h_2(\omega), h_3(\omega)$, and they become,

$$h_1 = -0.8321\xi_1 + 0.5547\xi_2 \leq 2.08013 \tag{9.18}$$

$$h_2 = 0.8944\xi_1 + 0.4472\xi_2 \leq 2.68328 \tag{9.19}$$

$$h_3 = -0.2425\xi_1 - 0.9701\xi_2 \leq 1.94029 \tag{9.20}$$

The values of coefficients in the Edgeworth series expansion can be obtained by calculating the corresponding moments (see chapter 7, or reference [117]). Since the polynomial chaos expansion is a function of normalized independent Gaussian distributed random variables, the computation of the moments becomes effortless [120]. For example the coefficient $a_{1,1}$ in the expansion equals to $m_1(h_1) = 0$ and $a_{1,2,1,1} = m_{1,1}(h_1, h_2) = -0.74423\xi_1^2 + 0.124009\xi_1\xi_2 + 0.248062\xi_2^2 = -0.496168$. For the case of an eleven-term Edgeworth series expansion, the corresponding contribution of each coefficient to the approximation of the $P(h_1 \leq 2.08013, h_2 \leq 2.68328, h_3 \leq 1.94029)$ is tabulated in table 9.5. It gives the stochastic flexibility index approximately of 0.952, and this value almost agrees with 10,000 sampling points from Monte Carlo method [163].

term	H_i	value
1	H_0	0.952079
2	$H_1(h_1)$	0
3	$H_1(h_2)$	0
4	$H_1(h_3)$	0
5	$H_2(h_1)$	0
6	$H_2(h_2)$	0
7	$H_2(h_3)$	0
8	$H_1(h_1)H_1(h_2)$	−0.000241471
9	$H_1(h_1)H_1(h_3)$	−0.000933092
10	$H_1(h_2)H_1(h_3)$	−0.000422747
11	$H_1(h_1)H_1(h_2)H_1(h_3)$	0

Table 9.5: Contribution of each term in the flexibility index calculation of example 4.

9.2.2 Nonlinear inequality model

A simple system of liquid transportation problem in chapter 6 is considered again in here. This example problem is taken from Grossmann and Floudas [59] and it will illustrate the nonlinear case of stochastic flexibility index problem. A set of reduced constraints for such a system is given by:

$$h_1 \equiv P_1 + \rho H - \epsilon - \frac{m^2}{\rho r^2 C_v^{max2}} - km^{1.84}D^{-5.16} - P_2' \leq 0 \tag{9.21}$$

$$h_2 \equiv -P_1 - \rho H - \epsilon + \frac{m^2}{\rho C_v^{max2}} + km^{1.84}D^{-5.16} + P_2' \leq 0 \tag{9.22}$$

$$h_3 \equiv -\epsilon \leq 0 \tag{9.23}$$

$$h_4 \equiv -0.5C_v^{max}(1-r) \leq 0 \tag{9.24}$$

To simplify the expectation calculations, all nonlinear forms should be transformed or approximated as polynomials. In this case, the Adomian polynomials [2] are used to approximate every nonlinear term in the inequality model above. The second order Adomian polynomials produce the following approximations:

$$\begin{aligned} \frac{1}{\rho} &= \frac{1}{E(\rho)} - \frac{\sigma_\rho}{E(\rho)^2}\xi_1 + \frac{{\sigma_\rho}^2}{E(\rho)^3}{\xi_1}^2 \\ m^{1.84} &= E(m)^{1.84} + 1.84\sigma_m E(m)^{0.84}\xi_2 + 0.7728{\sigma_m}^2 E(m)^{-0.16}{\xi_2}^2 \end{aligned}$$

where ξ_1, ξ_2 are independent standard Gaussian random variables.

Since the third and fourth constraints are always satisfied by such ranges of data, only the first two constraints will be considered. Substituting all required data and expansions, and normalizing the slack variables of the first two inequalities, we have

$$\begin{aligned} h_1 &= 0.103742 + 0.126653\xi_1 - 0.00602449\xi_1^2 - \\ &\quad 0.976829\xi_2 + 0.0481959\xi_1\xi_2 - 0.0024098\xi_1^2\xi_2 - \\ &\quad 0.0974763\xi_2^2 + 0.00481959\xi_1\xi_2^2 - 0.00024098\xi_1^2\xi_2^2 - \\ &\quad 0.00175421\xi_3 - 0.00064555\xi_2\xi_3 - \\ &\quad 0.0000542262\xi_2^2\xi_3 - 0.0474112\xi_4 \leq 2.49362 \\ h_2 &= -0.0266538 - 0.128476\xi_1 + 0.00029932\xi_1^2 + \\ &\quad 0.304479\xi_2 - 0.00239456\xi_1\xi_2 + 0.000119728\xi_1^2\xi_2 + \\ &\quad 0.0263425\xi_2^2 - 0.000239456\xi_1\xi_2^2 + \\ &\quad 0.0000119728\xi_1^2\xi_2^2 + 0.0348625\xi_3 + 0.0128294\xi_2\xi_3 \\ &\quad +0.00107767\xi_2^2\xi_3 + 0.94223\xi_4 \leq 0.324734 \end{aligned}$$

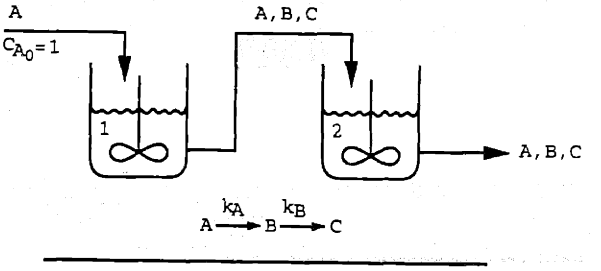
Similar to the previous example, the joint probability density function of these two random slack variables can be approximated by the Edgeworth series expansion. A two-term Edgeworth series expansion is used in this problem, and it first two coefficients a_0 and a_1 are equal to 1 and $\frac{-0.612953}{3!}$ respectively. Hence, the first term of cumulative density function approximation equals to 0.623343, and the second one is 0.00595569. This means that the stochastic flexibility index of this system is about 0.629299. Such a result is very close to 0.63000 that obtained from using 10000 sampling points of Hammersley-Wozniakowski

sampling method (see chapter 6).

9.3 Stochastic optimization examples

9.3.1 Reactors sequence problem

An optimization example of a reactor sequence design taken from Manousiothakis and Sourlas [101] is considered and modified to include a chance constraint subproblem as the stochastic flexibility index problem. It should be noted that the number of local maxima of the original problem is not two, as reported by Manousiothakis and Sourlas, but three, *i.e.*, $c_{B_2} = 0.38803, 0.38664, 0.37455$ mol/lt. The second local maximum is obtained when decision variables V_1 and V_2 are equal to 2.834 and 5.367, respectively. The sequence of reactors is shown in figure 9-26.



	Reactor 1	Reactor 2
k_A	0.096540	0.097515
k_B	0.035272	0.039191

Figure 9-26: Reactor sequence of optimization example.

The original deterministic formulation of the problem is written as follows:

$$
\begin{aligned}
\max_{V_1,V_2} \quad & c_{B_2} \\
\text{s.t.} \quad & c_{A_1} - c_{A_0} + k_{A_1} c_{A_1} V_1 = 0 \\
& c_{A_2} - c_{A_1} + k_{A_2} c_{A_2} V_2 = 0 \\
& c_{B_1} + c_{A_1} - c_{A_0} + k_{B_1} c_{B_1} V_1 = 0 \\
& c_{B_2} - c_{B_1} + c_{A_2} - c_{A_1} + k_{B_2} c_{B_2} V_2 = 0 \\
& z_1^2 - V_1 = 0 \\
& z_2^2 - V_2 = 0 \\
& z_1 + z_2 \leq 4
\end{aligned}
$$

Consider that the reaction rate constant k_{A_1} is uncertain and it follows a Gaussian distribution with mean value 0.096540 and standard deviation 0.03. The polynomial chaos expansion of k_{A_1} becomes,

$$k_{A_1} = k_{A_{10}} + k_{A_{11}} \varepsilon_1$$

The next step is to expand all state variables into polynomial chaos expansions,

$$
\begin{aligned}
c_{A_1} &= c_{A_{10}} + c_{A_{11}} \varepsilon_1 + c_{A_{12}} (\varepsilon_1^2 - 1) + \cdots \\
c_{A_2} &= c_{A_{20}} + c_{A_{21}} \varepsilon_1 + c_{A_{22}} (\varepsilon_1^2 - 1) + \cdots \\
c_{B_1} &= c_{B_{10}} + c_{B_{11}} \varepsilon_1 + c_{B_{12}} (\varepsilon_1^2 - 1) + \cdots
\end{aligned}
$$

$$c_{B_2} = c_{B_{20}} + c_{B_{21}}\varepsilon_1 + c_{B_{22}}(\varepsilon_1^2 - 1) + \cdots$$

The decision variables, V_1 and V_2, are chosen to be deterministic variables. Substituting the above expansions to the original constraints formulation and using variational approach to transform the stochastic constraints to their deterministic equivalent, the model becomes (see appendix F for DEMM input file of this example):

$$\begin{aligned}
&-c_{A_0} + c_{A_{10}} + c_{A_{10}}k_{A_{10}}V_1 + c_{A_{11}}k_{A_{11}}V_1 = 0\\
&c_{A_{11}} + c_{A_{11}}k_{A_{10}}V_1 + c_{A_{10}}k_{A_{11}}V_1 + 2c_{A_{12}}k_{A_{11}}V_1 = 0\\
&2c_{A_{12}} + 2c_{A_{12}}k_{A_{10}}V_1 + 2c_{A_{11}}k_{A_{11}}V_1 = 0\\
&-c_{A_0} + c_{A_{10}} + c_{B_{10}} + c_{B_{10}}k_{B_1}V_1 = 0\\
&c_{A_{11}} + c_{B_{11}} + c_{B_{11}}k_{B_1}V_1 = 0\\
&2c_{A_{12}} + 2c_{B_{12}} + 2c_{B_{12}}k_{B_1}V_1 = 0\\
&-c_{A_{10}} + c_{A_{20}} + c_{A_{20}}k_{A_2}V_2 = 0\\
&-c_{A_{11}} + c_{A_{21}} + c_{A_{21}}k_{A_2}V_2 = 0\\
&-2c_{A_{12}} + 2c_{A_{22}} + 2c_{A_{22}}k_{A_2}V_2 = 0\\
&-c_{A_{10}} + c_{A_{20}} - c_{B_{10}} + c_{B_{20}} + c_{B_{20}}k_{B_2}V_2 = 0\\
&-c_{A_{11}} + c_{A_{21}} - c_{B_{11}} + c_{B_{21}} + c_{B_{21}}k_{B_2}V_2 = 0\\
&-2c_{A_{12}} + 2c_{A_{22}} - 2c_{B_{12}} + 2c_{B_{22}}k_{B_2}V_2 = 0\\
&z_1^2 - V_1 = 0\\
&z_2^2 - V_2 = 0\\
&z_1 + z_2 \leq 4
\end{aligned}$$

An additional stochastic inequality is included to represent the flexibility of the system with respect to the uncertain value of reaction rate constant k_{A_1},

$$Pr\{c_{B_2} \geq \alpha\} \geq \beta \tag{9.25}$$

Using Edgeworth series expansion (equation (7.10)), the inequality (9.25) can be approximated as follows:

$$\begin{aligned}
z_3 &= \frac{c_{B_{20}} - \alpha}{\sqrt{c_{B_{21}}^2 + 2c_{B_{22}}^2}}\\
z_4 &= \frac{-6c_{B_{21}}^2 c_{B_{22}}^2 - 8c_{B_{22}}^3}{(c_{B_{21}}^2 + 2c_{B_{22}}^2)^{3/2}}\\
\beta &\leq 0.5 + 0.5Erf(z_3) - \frac{1}{\sqrt{2\pi}}e^{\frac{z_3^2}{2}}\frac{z_4}{3!}(z_3^2 - 1) + \cdots
\end{aligned}$$

Several alternatives to reformulate the objective functions in the DEMM environment are available (see chapter 7). The simplest one is to use the expected or mean value of c_{B_2}. Other alternatives are related to the Bayesian approach, which defines a utility function as the objective function, for example $E(c_{B_2}) + w_1\beta - w_2\alpha$, where w_1, w_2 are weighting factors. The value of these weighting factors might be changed as the decision maker gains more knowledge or information.

For this example, the value of α is set to 0.36 and w_2 to 0. Table 9.6 shows the results

from **GAMS** [21] as the value of the weighting factor w_1 changes. While the flexibility

w_1	β	$E(c_{B_2})$	V_1	V_2
0.0	0.88125	0.38321 mol/lt	3.90 s	4.11 s
0.025	0.89519	0.38303 mol/lt	3.32 s	4.75 s
0.05	0.91612	0.38222 mol/lt	2.59 s	5.71 s
0.075	0.98274	0.37801 mol/lt	1.07 s	8.79 s
0.1	0.99246	0.37719 mol/lt	0.86 s	9.43 s
1.0	0.99982	0.37583 mol/lt	0.52 s	10.76 s

Table 9.6: Solution of reactor sequence optimization example.

index is increasing, the result moves to the zero variance solution. It means that the volume of the first reactor becomes zero, and no uncertain parameter is involved. Within a fixed cost horizon represented by equation (9.25), the expected value of c_{B_2} decreases as the flexibility index, β, increases. Therefore, the decision maker has to consider the trade off between β and $E(c_{B_2})$. A different measure, such as the differential sensitivity of the $E(c_{B_2})$ to the flexibility index β, $\frac{dE(c_{B_2})}{d\beta}$, can then be used. This sensitivity index may aid the decision maker. For example, if $E(c_{B_2})$ is the most important objective, then the less the sensitivity index is the better the alternative will be, and vice versa.

9.3.2 Stochastic transportation problem

To illustrate the use of collocation method for solving a distribution problem of the stochastic optimization model, an example of stochastic transportation problem is considered here [13]. Table 9.7 below outlines the transportation costs from the warehouses, W_i $(1 \leq i \leq 3)$, to the consumption centers, C_j $(1 \leq j \leq 4)$. The last column denotes the maximum cumulative request that each warehouse may fulfill while the last line contains the consumptions of C_j. In this problem, we assume that the random consumptions u and v are independent random variables with exponential distributions Exp(0.002) and Exp(0.003) respectively. Therefore, this stochastic transportation problem can be formulated as

	C_1	C_2	C_3	C_4	
W_1	3	4	5	2	$\leq$ 11000.0
W_2	2	3	5	4	$\leq$ 13000.0
W_3	6	5	2	3	$\leq$ 8000.0
	$7000.0 + u$	10000.0	5000.0	$1000 + v$	

Table 9.7: Transportation costs from warehouses to consumption centers.

follows:

$$
\begin{aligned}
\min\ z \ = \ & 3x_{11} + 4x_{12} + 5x_{13} + 2x_{14} + 2x_{21} + 3x_{22} + \\
& 5x_{23} + 4x_{24} + 6x_{31} + 5x_{32} + 2x_{33} + 3x_{34} \\
\text{s.t.} \quad & \textstyle\sum_{j=1}^{5} x_{1j} = 11000.0 \qquad \sum_{i=1}^{3} x_{i2} = 10000.0 \\
& \textstyle\sum_{j=1}^{5} x_{2j} = 13000.0 \qquad \sum_{i=1}^{3} x_{i3} = 5000.0 \\
& \textstyle\sum_{j=1}^{5} x_{3j} = 8000.0 \qquad \sum_{i=1}^{3} x_{i4} = 1000.0 + v \\
& \textstyle\sum_{i=1}^{3} x_{i1} = 7000.0 + u \qquad \sum_{i=1}^{3} x_{i5} = 9000.0 - u - v.
\end{aligned}
$$

In this problem, we will use problem-specific polynomial chaos expansions to represent the uncertain parameters and the uncertain optimum value. Since the distributions of both uncertain parameters are exponential, we can use standard independent exponential random variables, ξ_1 and ξ_2 whose mean values and standard deviations equal to 1, to represent uncertain parameters,

$$
\begin{aligned}
u &= u_0 - u_1(1 - \xi_1) \\
v &= v_0 - v_1(1 - \xi_2),
\end{aligned}
$$

where $u_0 = 500.0, v_0 = 333.333$ and $u_1 = 500.0, v_1 = 333.333$ are their mean values and standard deviations respectively. To represent the uncertain optimum value, z, we will use Laguerre polynomials of exponential random variables, and it is approximated by using a five-term problem-specific polynomial chaos expansion,

$$
z = z_0 + z_1(1 - \xi_1) + z_2(1 - \xi_2) + z_3(\xi_1^2 - 4\xi_1 + 2) + z_4(\xi_2^2 - 4\xi_2 + 2). \quad (9.26)
$$

With such a representation, the mean value of z equals to the value of the first coefficient z_0 while the variance of z can be calculated according to the following equation,

$$
\sigma^2(z) = z_1^2 + z_2^2 + 4z_3^2 + 4z_4^2.
$$

Those five coefficients in that problem-specific polynomial chaos expansion are calculated

by treating the problem as a black-box type model. Consequently, we need to choose five orthogonal collocation points. The basis of collocation points in this case are the roots of the third order Laguerre polynomial, from which we have the following collocation points, (ξ_1, ξ_2),

$$\begin{aligned} p_1 &= (2.29428, 2.29428) \\ p_2 &= (0.415775, 2.29428) \\ p_3 &= (6.28995, 2.29428) \\ p_4 &= (2.29428, 0.415775) \\ p_5 &= (2.29428, 6.28995). \end{aligned}$$

We then solve the deterministic optimization problem at each collocation point (we use **GAMS/MINOS** to solve that linear optimization problem), as a result, we obtain the implicit deterministic equivalent equations for those coefficients,

$$\begin{aligned} z_0 - 1.29428z_1 - 1.29428z_2 - 1.9134z_3 - 1.9134z_4 &= 64970.94 \\ z_0 + 0.584225z_1 - 1.29428z_2 + 0.509769z_3 - 1.9134z_4 &= 62153.18 \\ z_0 - 5.28995z_1 - 1.29428z_2 + 16.4037z_3 - 1.9134z_4 &= 70964.43 \\ z_0 - 1.29428z_1 + 0.584225z_2 - 1.9134z_3 + 0.509769z_4 &= 63718.60 \\ z_0 - 1.29428z_1 - 5.28995z_2 - 1.9134z_3 + 16.4037z_4 &= 67634.72. \end{aligned}$$

The solution of these equations is given by

$$\begin{aligned} z_0 &= 62166.7 \\ z_1 &= -1500.0 \\ z_2 &= -666.668 \\ z_3 &= -0.000865638 \\ z_4 &= -0.000302078. \end{aligned}$$

Hence, the mean value of z is 62166.7, and its standard deviation equals to 1641.48. We can also simulate the value of z by using equation (9.26). The following commands are executed in the S language environment [11] to generate the empirical density function of z:

```
> x1 <- rexp(10000)
> x2 <- rexp(10000)
> z <- 62166.7 - 1500.0*(1.0-x1) - 0.000865638*(x1*x1-4*x1+2) -
+ 666.668*(1.0-x2) - 0.000302078*(x2*x2-4*x2+2)
> plot(density(z,n=200),xlab="z",ylab="f(z)",type="l")
```

and the result is shown in figure 9-27, which is very close to the result from the Laplace transform method [13] as well as from the Monte Carlo simulation method with 5000 trials [139]. From figure 9-27, we see that the mode of optimum value is around 61000.0. This suggests that the probability of having the value around 61000.0 as the optimal value is quite large. On the other hand, the probability that the optimal value will be less that 60000.0 and larger than 70000.0 is quite small.

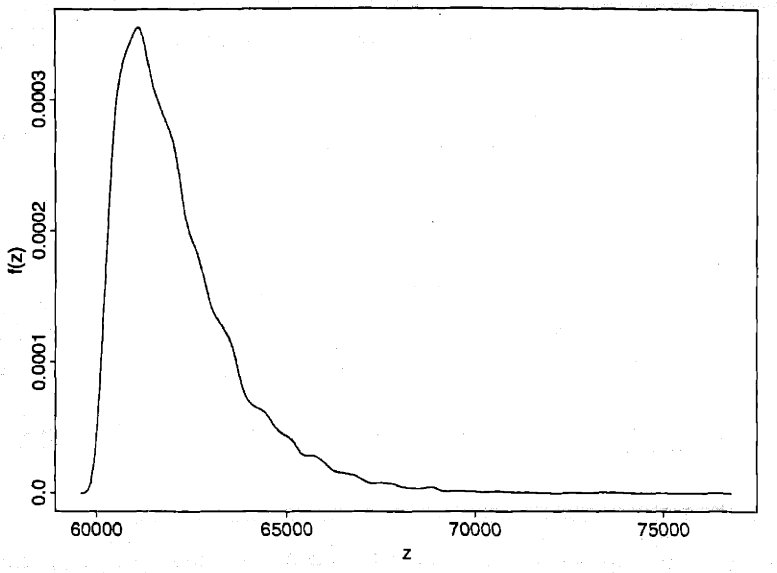


Figure 9-27: The probability density function of optimum value.

Table 9.8 summarizes all examples given in this chapter. One key point from this summary is that the results from DEMM are quite close to those of analytical or Monte Carlo methods. Another point is that the DEMM can be effectively applied to a wide range of problems with very different complexity.

Model	Number of state variables	Number of uncertain parameters	DEMM	Others approaches	Comments
Algebraic equations Gaussian	3	2	Galerkin $3 \times 6 =$ 18 eqns	Analytical	pdf from DEMM is very close to that of analytical result.
Algebraic equations Log-normal	3	9	Galerkin $3 \times 34 =$ 102 eqns	Analytical	pdf from DEMM is very close to that of analytical result.
Algebraic equations Gaussian	183	4	Galerkin 2147 eqns 24.34 secs Collocation 9 pts 16.49 secs	Monte Carlo 2741 pts 2182.56 secs	pdf from DEMM is very close to that of Monte Carlo result.
Differential equations Moments are known	3	2	Galerkin $3 \times 8 =$ 24 eqns		stability can change in the presence of uncertainty.
Differential equations Gaussian	12	12	Galerkin $12 \times 25 =$ 300 eqns 228.5 secs		DEMM provides both sensitivity and uncertainty analyses.
Differential equations Gaussian	7	10	Galerkin $7 \times 21 =$ 147 eqns 13.52 secs		sensitivity $\times$ uncertainty index brings more insight.
Inequality model linear Gaussian	3	2	Edgeworth 11 terms	Monte Carlo 10000 pts [163]	SF index from DEMM is close to that of Monte Carlo.
Inequality model non-linear Gaussian	4	2	Edgeworth and Adomian polynomials 2 and 3 terms	Monte Carlo 10000 pts	SF index from DEMM is close to that of Monte Carlo.

continued on next page

continued from previous page

Model	Number of state variables	Number of uncertain parameters	DEMM	Others approaches	Comments
Optimization model decision problem Gaussian	4	1	Galerkin and Edgeworth expansion $4 \times 3 =$ 12 eqns and 2 terms	Monte Carlo 1000 pts	Optimal solutions from DEMM is close to that of Monte Carlo.
Optimization model distribution problem Exponential	16	2	Collocation 5 pts	Monte Carlo 5000 pts [139]	Optimal pdf from DEMM is close to that of Monte Carlo.

Table 9.8: Summary of examples.

Chapter 10

Directions for Future Research

Several studies in the field of the uncertainty analysis of chemical and environmental engineering systems are needed in order to refine the deterministic equivalent modeling method. These studies can be classified into two domains. One is the advancement of theoretical and methodological foundations of the deterministic equivalent modeling method. Another is related to the improvement of the implementation of this proposed method.

10.1 Advancement of theoretical and methodological foundations

Critical theoretical foundations of the deterministic equivalent modeling method are the convergence and the stability of the solution. The mean-square convergence of polynomial chaos expansion suggests the L_2 type of convergence for the solution. However, the use of the Galerkin's approach and the collocation method in the probabilistic setting may contribute different errors in the transformation process.

In the case of the probabilistic collocation method, its convergence and stability follow the same result as of the deterministic collocation method because the formulation of the probabilistic collocation method and the deterministic collocation method are the same if the joint probability density function of basis random variables is not equal to zero for all chosen collocation points. In other words, the the probabilistic collocation method (see chapter 7),

$$\int_{\{\xi_i\}} f_{\{\xi_i\}}(\{\xi_i\})\, R_N(\mathbf{x}, \{\xi_i\})\, \delta(\{\xi_i\} - \{p_i\}_j)\, d\{\xi_i\} \;=\; 0; \;\; j = 1, \ldots, N,$$

yields the formulation,

$$R_N(\mathbf{x}, \{p_i\}_j) \;=\; 0; \;\; j = 1, \ldots, N,$$

which is similar to the formulation of the deterministic collocation method,

$$\int_{\{\xi_i\}} R_N(\mathbf{x}, \{\xi_i\})\, \delta(\{\xi_i\} - \{p_i\}_j)\, d\{\xi_i\} \;=\; 0; \;\; j = 1, \ldots, N,$$

if the values of joint probability density function of basis random variables $\{\xi_i\}$ are not

equal to zero for N chosen collocation points $\{p_i\}_j$, $j = 1, \ldots, N$,

$$f_{\{\xi_i\}}(\{p_i\}_j) \neq 0; \quad j = 1, \ldots, N.$$

Therefore, the result from the study of the convergence and the stability of deterministic collocation method can be applied to the case of the probabilistic collocation method. As an example, the stability condition described in the following lemma [26] for the solution of the deterministic linear operator problem, such as a boundary value problem,

$$\begin{aligned} Lu &= f \\ Bu &= 0, \end{aligned}$$

is also valid for the case of similar stochastic linear operator problems,

Lemma 10.1 *If there exists a constant $\bar{\alpha} > 0$ independent of N such that*

$$\bar{\alpha}\|u\|_W \leq \sup_{\substack{v \in Y_N \\ v \neq 0}} \frac{(L_N u, v)_N}{\|u\|_V}; \quad \forall u \in X_N$$

then the approximation is stable, in the sense that the estimate

$$\|u^N\|_W \leq C \, \|f\|_N$$

holds for a constant $C > 0$ independent of N.
W and V represent Hilbert spaces with their appropriate norms, and L_N is an approximation to the operator L in which derivatives are taken at the collocation points. X_N is the space of the polynomials of degree $\leq N$ which satisfy the boundary condition, and Y_N is the space spanned by the approximating polynomials.

Similarly, the convergence of the collocation method to the deterministic linear operator problem according to the following lemma [26]

Lemma 10.2 *The error of the solution can be estimated according to the following formula*

$$\|u - u^N\|_W \leq \left(1 + \frac{\beta}{\bar{\alpha}}\right) \|u - P_N u\|_W + \sup_{\substack{v \in Y_N \\ v \neq 0}} \frac{|(LP_N u, v) - (L_N P_N u, v)_N|}{\|v\|_V} + \sup_{\substack{v \in Y_N \\ v \neq 0}} \frac{(f, v) - (f, v)_N}{\|v\|_V},$$

where P_N denotes the projection operator from the space of sufficiently smooth functions upon the finite approximation space X_N.

is also valid for the stochastic linear operator problem.

Moreover, if the joint probability density function of basis random variables is not a function of those basis random variables (for example, the case of uniformly distributed basis random variables), then the stability and the convergence results of the Galerkin's approach for the deterministic problem will also be valid for similar stochastic problems. As

an example, the stability and the convergence of the deterministic linear operator problem which are given in the following lemmas [26]

Lemma 10.3 *If the following condition holds*

$$\alpha \|u\|_E^2 \leq (Lu, u); \quad \forall u \in X_N$$

then the Galerkin approximation is stable, in the sense that the estimate

$$\|u^N\|_E \leq C \|f\|$$

holds with a constant $C > 0$ independent of N.
E denotes a Hilbert space with "finite" energy, and the energy is precisely given by $\|u\|_E^2$.

Lemma 10.4 *The following error estimate*

$$\|u - u^N\|_E \leq \left(1 + \frac{\beta}{\alpha}\right) \|u - P_N u\|_E$$

implies the convergence of the Galerkin approximation.

are also valid for the case of the stochatic problems whose joint probability density function of basis random variables is, in fact, a constant. Since the case of non-constant joint probability density function of basis random variables has not been studied yet, the general stability and convergence results for the probabilistic Galerkin's approach are not available. Therefore, it would be useful to conduct such a study in the future in order to establish a more general theoretical foundation of the Galerkin's approach in the deterministic equivalent modeling method.

Another future research of the deterministic equivalent modeling method is the development of a methodology or a procedure for dealing with discrete random variables. This topic is important because many chemical and environmental engineering variables are natural discrete variables. As an example, the number of monomers in a polymer chain, the availability of equipment, or the availability of a specific technology for reducing a specific set of pollutants or recycling hazardous wastes.

For a small scale problem involving discrete random variables, one can enumerate all possible states or use a bounding scheme, such as the Chebyshev inequality [120], or problem-specific bounding procedure [163], to solve the problem. Clearly, for a problem with a large set of possible outcomes or states, the use of bounding scheme will not help. Therefore, the study of discrete polynomial chaos expansions [90] seems necessary in this case. Such a study may be further motivated by the success of applying discrete weighted residual methods, such as the Galerkin's approach and the collocation method, to polymerization reactions problem [25].

10.2 Improvement of current implementation

In the future, the implementation of the language of the deterministic equivalent modeling method should include all the results of the latest advancement in theoretical and methodological foundations of the method. This means that the language will be able to handle the case of continuous and discrete uncertain parameters as well as able to estimate the error of

approximations. Another important feature in uncertainty analysis that needs to be incorporated is the estimation of the probability density function of uncertain parameters from a given set of data. Therefore, methods described in chapter 3, such as the Bayesian approach and the minimum cross-entropy method, will become a part of future implementation. In other words, the whole spectrum of the process for performing uncertainty analysis will be in one language. Such a comprehensive language will definitely reduce the complexity of any activity involving uncertainty analysis.

To further reduce that complexity, the design of a dedicated and sophisticated user interface for the language of the deterministic equivalent modeling method becomes a critical issue. This user interface should provide an easy access not only to the language, but also to the collection of deterministic solvers for standard problems in the chemical and environmental engineering systems. Several key points should be considered in the development of such an interface:

Input to the language A robust graphical and hypertextual based interface should be made for this phase. It will help the user in the construction phase of the stochastic model of interest, as well as in defining the uncertainty of parameters or inputs to the model.

Processing phase A collection of deterministic solvers for standard problems should be available at the hand of the user, and the interface should be able to suggest various appropriate solvers based on the type of the stochastic model in consideration. Furthermore, a diagnostic tool should be made such that it can be invoked by the user on the fly to trace the process.

Output from the language A standard reporting tool should be made to process the results from solving a stochastic model. It should be able to produce and to show some standard text and image files based on the outputs. Moreover, there should be some options for the user to add specific features to those text and image files.

Another critical issue in the development of user interface to the language of the deterministic equivalent modeling method is the openness of its architecture. Since it is useful to the modelers in a certain field of chemical and environmental engineering to be able to produce a complete package of their models which provides the user of the package to perform uncertainty analysis, the user interface of the DEMM language should be portable, and should have a standard protocol for its inputs and outputs.

Chapter 11

Conclusions

After applying the proposed technique, the deterministic equivalent modeling method, to some chemical and environmental engineering problems described in the previous chapter, we may draw some general conclusions on the development and the use of such a method. The conclusions from developing and improving this method are given in the next subsection while the conclusions from applying it will be put in the other subsection.

11.1 Development of the deterministic equivalent modeling method

There are three key points that can be made from the development of a methodology for applying uncertainty analysis to a wide range of problems:

- A consistent and general uncertainty representation is critical and necessary.
- A notion of reusing the deterministic solvers for solving the corresponding stochastic models should be considered.
- Devising a systematic way to transform different stochastic models into their deterministic equivalent models becomes the main focus of the work.

Each key point will be discussed in the following text.

In chapter 4, we have discussed two types of representations for random variables. One type is the direct representations of random variables, and the other is the indirect representations. An important class of direct representations is the polynomial chaos expansion. Since this expansion is convergent in the mean-square sense [24], we can use it to closely approximate an arbitrary random variable such as the uncertain parameter or the uncertain response variable of the stochastic model. Furthermore, this research also proposes the use of problem-specific polynomial chaos expansion which not only simplifies the representation, but also improves its convergent property, especially for a random variable whose distribution is very different than the Gaussian. This problem-specific polynomial chaos expansion follows the notion of response surface function from the field of experimental design [18]. However, it extends the function to include high-order polynomials.

Another way to represent a random variable is by approximating it indirectly. Orthogonal series expansions, such as the Edgeworth series expansion, and the optimal combination of distributions technique are critical to the transformation process of stochastic inequality

models. The application of the optimal combination of distributions extends the orthogonal series expansions in the same fashion as the problem-specific polynomial chaos expansion does to the regular polynomial chaos expansion. Together with the polynomial chaos expansion, these indirect representations can cover either recourse or chance-constrained formulations of stochastic inequality models. Since we choose to use the probabilistic representation as the common ground for describing uncertain parameters and variables, the polynomial chaos expansion and the optimal combination of distributions technique consequently become the crucial elements in producing a consistent and general representation of random variables.

Nevertheless, it should be pointed out that there are limitations associated with these direct and indirect representations. One obvious limitation is that when the uncertain response variable is not a continuous function of the uncertain parameters, the polynomial chaos expansion may find difficulties to represent that uncertain response variable. In other words, if the response function is very irregular, finite polynomial approximations may be insufficient as well as inaccurate to represent such a response function. Another difficulty is the approximation the probability distribution of highly correlated slack variables of the stochastic inequality model, because a large number of terms is required to produce an accurate approximation.

The second key point above mentions the consideration for applying the existing robust deterministic solvers to stochastic problems. In practice, both the availability of symbolic manipulation programs which are capable to handle large scale problems, and the application of reliable compiler generators may provide an environment where the notion of reusing the deterministic solvers to the stochastic models can be implemented. We have developed and implemented a prototype of the deterministic equivalent modeling method language, and it is described in chapter 8. Once we have a program to solve the corresponding deterministic model, it is relatively easy and straightforward in this language to define and to solve the stochastic model of interest.

Reusing the deterministic solvers for solving stochastic models requires the transformation of a stochastic model into its deterministic equivalent model. In other words, a systematic way to transform a wide range of stochastic models should be devised. To produce such a systematic way, an extension of the deterministic variational approach to the probabilistic space is studied. Both the Galerkin's approach and the collocation method have been investigated, and as a result, they actually can be applied to various classes of stochastic models: from stochastic algebraic and differential equations models to black-box process simulation problems and the distribution problem of stochastic optimization models. Therefore, one substantial product of the development phase of the deterministic equivalent modeling method is the prototype language for performing uncertainty analysis to various chemical and environmental engineering models in different programming languages or environments.

11.2 Application of the deterministic equivalent modeling method

In chapter 9, we have considered nine examples which represent typical chemical and environmental engineering models. While specific conclusions for each example are given in that chapter, in this subsection we wish to draw some general conclusions from those different examples. Several important points can be made from applying the deterministic equivalent

modeling method to those problems:

- Uncertainty analysis of a given model can be performed as an extension of the deterministic analysis.
- Sensitivity analysis should be combined with uncertainty analysis to provide a better understanding of the characteristics of the model.
- While the Galerkin's approach can provide more accurate moments values of response variables, the collocation method may produce more accurate cumulative distribution values as well as can handle black-box type models.

The generation of the deterministic equivalent model allows us to perform uncertainty analysis as an extension of the regular deterministic analysis. As a result, we can use the same deterministic solvers since the structure of the model is preserved. Furthermore, the prototype language of the deterministic equivalent modeling method simplifies the uncertainty analysis process. Such a language environment will foster the development of chemical and environmental engineering models which have uncertainty analysis features. People now can have a deterministic equivalent model instead of the deterministic model, and they can still use it as the deterministic model if they want to by setting all uncertainty factors in the model to be zero.

Figure 11-1 shows the need of combined sensitivity and uncertainty analysis. In practice, the values of some parameters may be unknown exactly, and the result from sensitivity analysis may suggest us to put more effort in studying the most sensitive parameter. However, this suggestion may not be optimal for reducing uncertainty of the response variable in the case that there are other less sensitive parameters but with much larger uncertainties, such as the case (b) in figure 11-1.

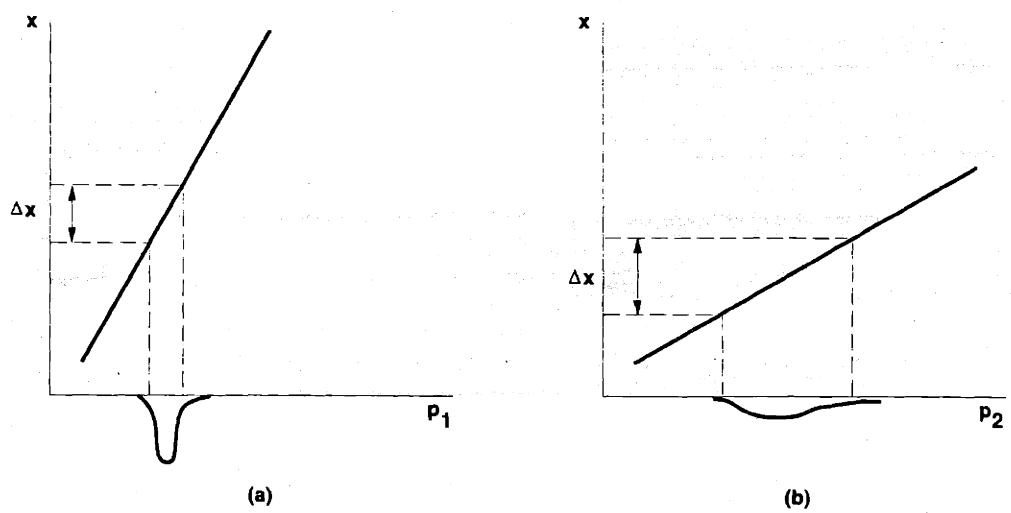


Figure 11-1: Two different cases of combined sensitivity and uncertainty analysis. Case (a) has high sensitivity but low uncertainty; case (b) has low sensitivity but high uncertainty.

Consequently, another measure should be used in this case. As an example, we can use a product of sensitivity index and standard deviation of uncertain parameter as a measure to

rank the contribution of each uncertain parameter to the uncertainty of response variables. This product measure is, in fact, directly available from the coefficients of the polynomial chaos expansion of a response variable.

Since collocation method can provide both the explicit and implicit deterministic equivalent models, the application area of this method is wider than that of the Galerkin's approach. As an example, the collocation method can deal with nonlinear non-polynomial models. Moreover, the collocation method extends the use of the deterministic equivalent modeling method to black-box type models. It also means that we can apply the deterministic equivalent modeling method for performing the uncertainty analysis to numerous existing models and their associated subroutines or programs.

The most important point in this conclusion is that the complexity of performing uncertainty analysis can be largely reduced through the use of DEMM. A reduction of 2 to 3 orders of magnitude for computational time is attainable (see table 11.1). Furthermore, the concept of reusing the deterministic solvers simplifies the analysis.

There is no question that uncertainty is always there in real systems. To cope with such an uncertainty, one can apply the DEMM – a computationally efficient methodology which allows direct incorporation of uncertainty in the model formulation.

Model type	DEMM (unit time)	Monte Carlo methods (unit time)
Equality model	1.0 – 1.6	304.6
Inequality model	1.0 – 1.6	90.9
Optimization model	1.0 – 1.8	200.0 – 1000.0

Table 11.1: Normalized average computational time from examples in chapter 9.

Appendix A

Simulated Annealing Result

The following text is the print-out of result from applying the "simulated annealing" algorithm [33] to the bayesian parameter estimation problem given in chapter 3.

```
SIMULATED ANNEALING EXAMPLE

NUMBER OF PARAMETERS:   2   MAXIMAZATION:     T
INITIAL TEMP:  1.0       RT: 0.50       EPS: 0.10E-05
NS:  20   NT:  5   NEPS:  4
MAXEVL:      10000   IPRINT: 1   ISEED1:    1   ISEED2:    2

                        STARTING VALUES
0.50000      0.50000

                        INITIAL STEP LENGTH
0.50000      0.50000

                        LOWER BOUND
0.00000E+00  0.00000E+00

                        UPPER BOUND
 1.0000       1.0000

                        C VECTOR
 2.0000       2.0000

***** INTERMEDIATE RESULTS FROM SA FOLLOW *****


                        INITIAL X
0.50000      0.50000
 INITIAL F:   4395049608332665.00


INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTION

 CURRENT TEMPERATURE:              1.0000
 MAX FUNCTION VALUE SO FAR:   0.20794701140225053OE+35
 TOTAL MOVES:                    200
    UPHILL:                       33
    ACCEPTED DOWNHILL:             0
    REJECTED DOWNHILL:           167
 OUT OF BOUNDS TRIALS:             8
```

```
 NEW MAXIMA THIS TEMPERATURE:       33

                         CURRENT OPTIMAL X
 0.20590       0.50028

                         STEP LENGTH (VM)
 0.76960E-02  0.16931E-01


 INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTION

  CURRENT TEMPERATURE:              0.50000
  MAX FUNCTION VALUE SO FAR:   0.216818721613493260E+35
  TOTAL MOVES:                      200
     UPHILL:                         31
     ACCEPTED DOWNHILL:               0
     REJECTED DOWNHILL:             169
  OUT OF BOUNDS TRIALS:               0
  NEW MAXIMA THIS TEMPERATURE:       31

                         CURRENT OPTIMAL X
 0.20740       0.49620

                         STEP LENGTH (VM)
 0.13682E-03  0.44224E-03


 INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTION

  CURRENT TEMPERATURE:              0.25000
  MAX FUNCTION VALUE SO FAR:   0.216881107071495620E+35
  TOTAL MOVES:                      200
     UPHILL:                         20
     ACCEPTED DOWNHILL:               0
     REJECTED DOWNHILL:             180
  OUT OF BOUNDS TRIALS:               0
  NEW MAXIMA THIS TEMPERATURE:       20

                         CURRENT OPTIMAL X
 0.20757       0.49545

                         STEP LENGTH (VM)
 0.11056E-05  0.65517E-05


 INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTION

  CURRENT TEMPERATURE:              0.12500
  MAX FUNCTION VALUE SO FAR:   0.216881107572268490E+35
  TOTAL MOVES:                      200
     UPHILL:                         25
     ACCEPTED DOWNHILL:               0
     REJECTED DOWNHILL:             175
  OUT OF BOUNDS TRIALS:               0
  NEW MAXIMA THIS TEMPERATURE:       12

                         CURRENT OPTIMAL X
 0.20757       0.49544
```

```
                        STEP LENGTH (VM)
0.23824E-07  0.70591E-07


INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTION

 CURRENT TEMPERATURE:              0.62500E-01
 MAX FUNCTION VALUE SO FAR:   0.216881107572268490E+35
 TOTAL MOVES:                      200
    UPHILL:                         89
    ACCEPTED DOWNHILL:               0
    REJECTED DOWNHILL:             111
 OUT OF BOUNDS TRIALS:               0
 NEW MAXIMA THIS TEMPERATURE:        0

                        CURRENT OPTIMAL X
0.20757      0.49544

                        STEP LENGTH (VM)
0.13236E-07  0.31374E-07


INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTION

 CURRENT TEMPERATURE:              0.31250E-01
 MAX FUNCTION VALUE SO FAR:   0.216881107572268490E+35
 TOTAL MOVES:                      200
    UPHILL:                         99
    ACCEPTED DOWNHILL:               0
    REJECTED DOWNHILL:             101
 OUT OF BOUNDS TRIALS:               0
 NEW MAXIMA THIS TEMPERATURE:        0

                        CURRENT OPTIMAL X
0.20757      0.49544

                        STEP LENGTH (VM)
0.20267E-07  0.31374E-07


INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTION

 CURRENT TEMPERATURE:              0.15625E-01
 MAX FUNCTION VALUE SO FAR:   0.216881107572268490E+35
 TOTAL MOVES:                      200
    UPHILL:                         90
    ACCEPTED DOWNHILL:               0
    REJECTED DOWNHILL:             110
 OUT OF BOUNDS TRIALS:               0
 NEW MAXIMA THIS TEMPERATURE:        0

                        CURRENT OPTIMAL X
0.20757      0.49544

                        STEP LENGTH (VM)
0.12667E-07  0.24402E-07


SA ACHIEVED TERMINATION CRITERIA. IER = 0.
```

***** RESULTS AFTER SA *****

SOLUTION

0.20757 0.49544

FINAL STEP LENGTH

0.12667E-07 0.24402E-07

OPTIMAL FUNCTION VALUE: 0.2168811075723E+35
NUMBER OF FUNCTION EVALUATIONS: 1401
NUMBER OF ACCEPTED EVALUATIONS: 387
NUMBER OF OUT OF BOUND EVALUATIONS: 8
FINAL TEMP: 0.1562500000000E-01 IER: 0

Appendix B

Bayesian Problem Subroutines

The following codes are used to solve the bayesian parameter estimation problem given in chapter 3. Subroutines **SA**, **EXPREP**, **RMARIN**, **RANMAR**, **PRT1** to **PRT10**, and **PRTVEC** are obtained from reference [33].

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "utility.c"
#include "ludcmp.c"

double poster(int n, int m, int p, double *tu, double *yinit,
              double *param, double **ymeas)
{

void fev_();
double  *par,*tuf,*yu0,**ymodel,*ymf,**v;
double d;
int i,j,k,index;
int *indx;

/* translate vector and matrix needed */
/* so they are suitable for FORTRAN */

par = dvector(0,p-1);
for (i = 1; i <= p; i++)
   { par[i-1] = param[i]; }

tuf = dvector(0,n-1);
for (i = 1; i <= n; i++)
   { tuf[i-1] = tu[i]; }

yu0 = dvector(0,m-1);
        for (i = 1; i <= m; i++)
     { yu0[i-1] = yinit[i]; }

        ymf = (double *)malloc((unsigned) n * m * sizeof(double));

/* calculate model for different times using LSODE */

        (void)fev_(&n,tuf,yu0,par,ymf);

/* transpose the model matrix back to c language standard */
```

```
      ymodel = dmatrix(0,n-1,0,m-1);
        for (i=0;i<=n-1;i++)
           { for (j=0;j<=m-1;j++)
                { index = j*n + i;
          ymodel[i][j] = ymf[index];
}
   }

/* start specific calculation for posterior distribution */
        /* calculate dispersion matrix v */

v = dmatrix(1,m,1,m);
for (i = 1; i <= m; i++)
           { for (j = 1; j <= m; j++)
              { v[i][j] = 0.0;
  for (k = 1; k <= n; k++)
     { v[i][j] += (ymeas[k][i] - ymodel[k-1][i-1])*
                                  (ymeas[k][j] - ymodel[k-1][j-1]);
     }
}
   }

/* calculate determinant of dispersion matrix v */
        /* we use LU decomposition technique from */
      /* NUMERICAL RECIPES book */

indx = ivector(1,m);
ludcmp(v,m,indx,&d);
        for (i = 1; i <= m; i++)
           { d *= v[i][i]; }

      /* now return the proportional value of posterior */

free_dvector(par,1);
    free_dvector(tuf,1);
free_dvector(ymf,0);
free_dvector(yu0,1);
free_dmatrix(ymodel,0,n-1,0);
free_dmatrix(v,1,m,1);
free_ivector(indx,1);

return (pow(d,-0.5*n));
}

double sample(int npoints, double xleft, double xright,
      double (*func)(int, int, int, double *, double *,
      double *, double **),
              int n, int m, int p, double *tu, double *yinit,
              double *param, double **ymeas, int pick, double size)
{

double *x,*par,*w,*f;
double dx,sumw,sumf,u,uinv;
int i;

/* discretize 1-D domain */
/* take middle point for each segment */

x = dvector(1,npoints);
```

```
        dx = (xright - xleft)/npoints;
 for (i = 1; i <= npoints; i++)
    { x[i] = xleft + (i-0.5)*dx; }

 /* generate weight to each segment */

 par = dvector(1,p);
 w = dvector(1,npoints);
 for (i = 1; i <= p; i++)
    { par[i] = param[i]; }
        sumw = 0.0;
 for (i = 1; i <= npoints; i++)
    { par[pick] = x[i];
      w[i] = func(n,m,p,tu,yinit,par,ymeas);
      sumw += w[i]*dx;
    }

 /* normalize the weight */

 f = dvector(1,npoints);
 for (i = 1; i <= npoints; i++)
    { w[i] = w[i]/sumw;
      f[i] = w[i]*dx;
    }

 /* generate uniform variate and inverse it */
 /* see RUBINSTEIN, pp. 41-42 */

 u = (double) (random()/size);
        i = 0;
        sumf = 0.0;
        while (sumf <= u)
           { i += 1;
       sumf += f[i];
     }
uinv = x[i] - 0.5*dx + (u - (sumf - f[i]))/w[i];

/* now deliver the sample */

free_dvector(x,1);
free_dvector(par,1);
free_dvector(w,1);
free_dvector(f,1);

return (uinv);
}

void griddy_gibbs_sampler(int steps, int p, double *parinit,
  int *npts, double *left, double *right,
  double size,
                double (*func)(int, int, int, double *, double *,
                               double *, double **),
                       int n, int m, double *tu, double *yinit,
                       double **ymeas, double **gibbs)
{

double *param;
int i,j;
```

```
/* calculate gibbs sampler for "steps" iterations */

param = dvector(1,p);
for (i = 1; i <= p; i++)
   { param[i] = parinit[i]; }
for (i = 1; i <= steps; i++)
   { for (j = 1; j <= p; j++)
{ param[j] = sample(npts[j],left[j],right[j],func,n,
                          m,p,tu,yinit,param,ymeas,j,size);
  gibbs[i][j] = param[j];
}
   }

free_dvector(param,1);

}

double *yinit,*param,**ymeas;
double *left,*right;
double *tu;
double post;
int    *npts;
int    i,j,k,n,m,p,steps,nofpoints;
double size;
double **spoints;
double tspoints[1000][30][2];
char   *input_file,*output_file;
char   name[256];
FILE   *fileread,*filewrite;


void databox()
{
     /* n = number of measurements */
     /* m = number of responses */
     /* p = number of unknown parameters */
     /* steps = number of iterations for gibbs sampler */

     n = 12; m = 3; p = 2; steps = 30;

     /* yinit = initial conditions */
     /* ymeas = measurement values of responses */
     /* param = vector of unknown parameters */
     /* tu = measurement times or independent variables */

     yinit = dvector(1,m);
     ymeas = dmatrix(1,n,1,m);
     param = dvector(1,p);
     tu = dvector(1,n);
     yinit[1] = 1.0;
     yinit[2] = 0.0;
     yinit[3] = 0.0;
     ymeas[1][1] = 0.959; ymeas[1][2] = 0.025; ymeas[1][3] = 0.028;
     ymeas[2][1] = 0.914; ymeas[2][2] = 0.061; ymeas[2][3] = 0.000;
     ymeas[3][1] = 0.855; ymeas[3][2] = 0.152; ymeas[3][3] = 0.068;
     ymeas[4][1] = 0.785; ymeas[4][2] = 0.197; ymeas[4][3] = 0.096;
     ymeas[5][1] = 0.628; ymeas[5][2] = 0.130; ymeas[5][3] = 0.090;
     ymeas[6][1] = 0.617; ymeas[6][2] = 0.249; ymeas[6][3] = 0.118;
     ymeas[7][1] = 0.480; ymeas[7][2] = 0.184; ymeas[7][3] = 0.374;
```

```
     ymeas[8][1] = 0.423; ymeas[8][2] = 0.298; ymeas[8][3] = 0.358;
     ymeas[9][1] = 0.166; ymeas[9][2] = 0.147; ymeas[9][3] = 0.651;
     ymeas[10][1] = 0.205; ymeas[10][2] = 0.050; ymeas[10][3] = 0.684;
     ymeas[11][1] = 0.034; ymeas[11][2] = 0.000; ymeas[11][3] = 0.899;
     ymeas[12][1] = 0.054; ymeas[12][2] = 0.047; ymeas[12][3] = 0.991;
     tu[1] = 0.5; tu[2] = 0.5;
     tu[3] = 1.0; tu[4] = 1.0;
     tu[5] = 2.0; tu[6] = 2.0;
     tu[7] = 4.0; tu[8] = 4.0;
     tu[9] = 8.0; tu[10] = 8.0;
     tu[11] = 16.0; tu[12] = 16.0;

     /* npts = number of points used for taking a sample using */
     /*        griddy-gibbs sampling technique */
     /* left, right = domain of parameters */

     npts = ivector(1,p);
     left = dvector(1,p);
     right = dvector(1,p);
     npts[1] = 20; npts[2] = 20;
     left[1] = 0.0; left[2] = 0.0;
     right[1] = 1.0; right[2] = 1.0;
}

void main(int argc,char *argv[])
{

     if (argc >= 4)
       { input_file = argv[1];
 output_file = argv[2];
 nofpoints = atoi(argv[3]);
       }
     else
       { printf("\nuse: box_draperc input output #points\n");
         nerror("Supply the required names of file and data");
       }

     /* open input file */

     if ((fileread=fopen(input_file,"rt")) == NULL)
       { nerror("Cannot read input file"); }

     srandom(1);
     size = pow(2.0,31.0)-1.0;
     databox();

     spoints = dmatrix(1,steps,1,p);
     for (i = 1; i <= nofpoints; i++)
        { for (j = 1; j <= p; j++)
     { fscanf(fileread,"%lf",&param[j]); }
          griddy_gibbs_sampler(steps,p,param,npts,left,right,size,poster,n,m,
               tu,yinit,ymeas,spoints);
  for (j = 1; j <= steps; j++)
     { for (k = 1; k <= p; k++)
  { tspoints[i][j][k] = spoints[j][k]; }
     }
        }

     fclose(fileread);
```

```
     /* open output file for every iteration */

     for (i = 1; i <= steps; i++)
        { sprintf(name,"%s.%02d",output_file,i);
          if ((filewrite=fopen(name,"wt")) == NULL)
            { nerror("Cannot write output file"); }
          for (j = 1; j <= nofpoints; j++)
            { for (k = 1; k <= p; k++)
  { fprintf(filewrite,"%12.10f  ",tspoints[j][i][k]); }
              fprintf(filewrite,"\n");
     }
          fclose(filewrite);
 }

}

C model box-draper
C
C the following is a simple model, with the coding
C needed for its solution by lsode.  the problem is from chemical
C kinetics, and consists of the following three rate equations..
C     dy1/dt = -param1*y1
C     dy2/dt = param1*y1 - param2*y2
C     dy3/dt = param2*y2
C on the interval from t = 0.0 to t = tfinal, with initial conditions
C y1 = yinit1, y2 = yinit2, y3 = yinit3.
C the following coding solves this problem with lsode, using mf = 22
C
C     program box-draper
C
      subroutine fev(nmeas, tfinal, yinit, param, ymod)
      double precision tfinal, yinit, param, ymod
      integer nmeas
      common double precision /par/ p
      dimension tfinal(12), yinit(3), param(2), p(2)
      dimension ymod(12,3)
      external fex
      double precision atol, rtol, rwork, t, tout, y, jdum
      dimension y(3), atol(3), rwork(58), iwork(23)
      neq = 3
      y(1) = yinit(1)
      y(2) = yinit(2)
      y(3) = yinit(3)
      p(1) = param(1)
      p(2) = param(2)
      t = 0.d0
      itol = 2
      rtol = 1.d-8
      atol(1) = 1.d-8
      atol(2) = 1.d-8
      atol(3) = 1.d-8
      itask = 1
      istate = 1
      iopt = 0
      lrw = 58
      liw = 23
      mf = 22
      do 40 iout = 1,nmeas
```

```
      tout = tfinal(iout)
      call lsode(fex,neq,y,t,tout,itol,rtol,atol,itask,istate,
     1           iopt,rwork,lrw,iwork,liw,jdum,mf)
      ymod(iout,1) = y(1)
      ymod(iout,2) = y(2)
      ymod(iout,3) = y(3)
   40 if (istate .lt. 0) go to 80
      return
   80 write(6,90)istate
   90 format(///22h error halt.. istate =,i3)
      return
      end

      subroutine fex (neq, t, y, ydot)
      double precision t, y, ydot
      dimension y(3), ydot(3)
      common double precision /par/ p
      dimension p(2)
      ydot(1) = -p(1)*y(1)
      ydot(2) = p(1)*y(1) - p(2)*y(2)
      ydot(3) = p(2)*y(2)
      return
      end

      PROGRAM SIMANN
C  This file is an example of the Corana et al. simulated annealing
C  algorithm for multimodal and robust optimization as implemented
C  and modified by by Goffe et al. An example from Box-Draper model (FCN)
C  is tried.

      PARAMETER (N = 2, NEPS = 4)

      DOUBLE PRECISION  LB(N), UB(N), X(N), XOPT(N), C(N), VM(N),
     1                  FSTAR(NEPS), XP(N), T, EPS, RT, FOPT

      INTEGER  NACP(N), NS, NT, NFCNEV, IER, ISEED1, ISEED2,
     1         MAXEVL, IPRINT, NACC, NOBDS

      LOGICAL  MAX

      EXTERNAL FCN

C  Set underflows to zero on IBM mainframes.
C     CALL XUFLOW(0)

C  Set input parameters.
      MAX = .TRUE.
      EPS = 1.0D-6
      RT = 0.5
      ISEED1 = 1
      ISEED2 = 2
      NS = 20
      NT = 5
      MAXEVL = 10000
      IPRINT = 1
      DO 10, I = 1, N
         LB(I) = 0.0D0
         UB(I) = 1.0D0
         C(I) = 2.0
```

```
10    CONTINUE

C  Note start at middle point of the domain.
      X(1) = 0.5
      X(2) = 0.5

C  Set input values of the input/output parameters.
      T = 1.0
      DO 20, I = 1, N
         VM(I) = 0.5
 20   CONTINUE

      WRITE(*,1000) N, MAX, T, RT, EPS, NS, NT, NEPS, MAXEVL, IPRINT,
     1              ISEED1, ISEED2

      CALL PRTVEC(X,N,'STARTING VALUES')
      CALL PRTVEC(VM,N,'INITIAL STEP LENGTH')
      CALL PRTVEC(LB,N,'LOWER BOUND')
      CALL PRTVEC(UB,N,'UPPER BOUND')
      CALL PRTVEC(C,N,'C VECTOR')
      WRITE(*,'('' '')')
      WRITE(*,'('' ***** INTERMEDIATE RESULTS FROM SA FOLLOW *****'')')

      CALL SA(N,X,MAX,RT,EPS,NS,NT,NEPS,MAXEVL,LB,UB,C,IPRINT,ISEED1,
     1        ISEED2,T,VM,XOPT,FOPT,NACC,NFCNEV,NOBDS,IER,
     2        FSTAR,XP,NACP)

      WRITE(*,'('' '')')
      WRITE(*,'('' ***** RESULTS AFTER SA *****   '')')
      CALL PRTVEC(XOPT,N,'SOLUTION')
      CALL PRTVEC(VM,N,'FINAL STEP LENGTH')
      WRITE(*,1001) FOPT, NFCNEV, NACC, NOBDS, T, IER

1000  FORMAT(/,' SIMULATED ANNEALING EXAMPLE',/,
     1       /,' NUMBER OF PARAMETERS: ',I3,'   MAXIMAZATION: ',L5,
     2       /,' INITIAL TEMP: ', G8.2, '   RT: ',G8.2, '   EPS: ',G8.2,
     3       /,' NS: ',I3, '   NT: ',I2, '   NEPS: ',I2,
     4       /,' MAXEVL: ',I10, '   IPRINT: ',I1, '   ISEED1: ',I4,
     5       '   ISEED2: ',I4)
1001  FORMAT(/,' OPTIMAL FUNCTION VALUE: ',G20.13
     1       /,' NUMBER OF FUNCTION EVALUATIONS:     ',I10,
     2       /,' NUMBER OF ACCEPTED EVALUATIONS:     ',I10,
     3       /,' NUMBER OF OUT OF BOUND EVALUATIONS: ',I10,
     4       /,' FINAL TEMP: ', G20.13,'  IER: ', I3)

      STOP
      END

      SUBROUTINE FCN(N,PAR,H)
C  Box-Draper model.

      DOUBLE PRECISION PAR(2), H
      INTEGER N

      DOUBLE PRECISION YINIT(3), YMEAS(12,3), TU(12)
      DOUBLE PRECISION YMF(12,3)

      DOUBLE PRECISION DET, DMAT(3,3)
      PARAMETER(NP=3)
```

```
      INTEGER I,J,K,INDX(3)

      EXTERNAL FEV, LUDCMP

      YINIT(1) = 1.0
      YINIT(2) = 0.0
      YINIT(3) = 0.0
      YMEAS(1,1) = 0.959
      YMEAS(1,2) = 0.025
      YMEAS(1,3) = 0.028
      YMEAS(2,1) = 0.914
      YMEAS(2,2) = 0.061
      YMEAS(2,3) = 0.000
      YMEAS(3,1) = 0.855
      YMEAS(3,2) = 0.152
      YMEAS(3,3) = 0.068
      YMEAS(4,1) = 0.785
      YMEAS(4,2) = 0.197
      YMEAS(4,3) = 0.096
      YMEAS(5,1) = 0.628
      YMEAS(5,2) = 0.130
      YMEAS(5,3) = 0.090
      YMEAS(6,1) = 0.617
      YMEAS(6,2) = 0.249
      YMEAS(6,3) = 0.118
      YMEAS(7,1) = 0.480
      YMEAS(7,2) = 0.184
      YMEAS(7,3) = 0.374
      YMEAS(8,1) = 0.423
      YMEAS(8,2) = 0.298
      YMEAS(8,3) = 0.358
      YMEAS(9,1) = 0.166
      YMEAS(9,2) = 0.147
      YMEAS(9,3) = 0.651
      YMEAS(10,1) = 0.205
      YMEAS(10,2) = 0.050
      YMEAS(10,3) = 0.684
      YMEAS(11,1) = 0.034
      YMEAS(11,2) = 0.000
      YMEAS(11,3) = 0.899
      YMEAS(12,1) = 0.054
      YMEAS(12,2) = 0.047
      YMEAS(12,3) = 0.991
      TU(1) = 0.5
      TU(2) = 0.5
      TU(3) = 1.0
      TU(4) = 1.0
      TU(5) = 2.0
      TU(6) = 2.0
      TU(7) = 4.0
      TU(8) = 4.0
      TU(9) = 8.0
      TU(10) = 8.0
      TU(11) = 16.0
      TU(12) = 16.0

      CALL FEV(12,TU,YINIT,PAR,YMF)
```

```
      DO 103 I=1,3
      DO 102 J=1,3
         DMAT(I,J) = 0.0
   DO 101 K=1,12
            DMAT(I,J) = DMAT(I,J) + (YMEAS(K,I) - YMF(K,I))*
     1                              (YMEAS(K,J) - YMF(K,J))
    101    CONTINUE
    102    CONTINUE
    103 CONTINUE

      CALL LUDCMP(DMAT,NP,NP,INDX,DET)
      DO 104 I=1,3
    104    DET = DET*DMAT(I,I)

      H = DET**(-0.5*12.0D0)

      RETURN
      END

      SUBROUTINE SA(N,X,MAX,RT,EPS,NS,NT,NEPS,MAXEVL,LB,UB,C,IPRINT,
     1              ISEED1,ISEED2,T,VM,XOPT,FOPT,NACC,NFCNEV,NOBDS,IER,
     2              FSTAR,XP,NACP)

C  Version: 3.1
C  Date: 6/10/91.
C  Differences compared to Version 2.0:
C     1. If a trial is out of bounds, a point is randomly selected
C        from LB(i) to UB(i). Unlike in version 2.0, this trial is
C        evaluated and is counted in acceptances and rejections.
C        All corresponding documentation is changed as well.
C  Differences compared to Version 3.0:
C     1. If VM(i) > (UB(i) - LB(i)), VM is set to UB(i) - LB(i).
C        The idea is that if T is high relative to LB & UB, most
C        points will be accepted, causing VM to rise. But, in this
C        situation, VM has little meaning; particularly if VM is
C        larger than the acceptable region. Setting VM to this size
C        still allows all parts of the allowable region to be selected.
C  Call to SA changed?
C     Yes. Integer work array IWK2 has been removed. In addition,
C          IWK1 has been renamed IWK.
C
C  Synopsis:
C  This routine implements the continuous simulated annealing global
C  optimization algorithm described in Corana et al.'s article
C  "Minimizing Multimodal Functions of Continuous Variables with the
C  "Simulated Annealing" Algorithm" in the September 1987 (vol. 13,
C  no. 3, pp. 262-280) issue of the ACM Transactions on Mathematical
C  Software.
C
C  A very quick (perhaps too quick) overview of SA:
C     SA tries to find the global optimum of an N dimensional function.
C  It moves both up and downhill and as the optimization process
C  proceeds, it focuses on the most promising area.
C     To start, it randomly chooses a trial point within the step length
C  VM (a vector of length N) of the user selected starting point. The
C  function is evaluated at this trial point and its value is compared
C  to its value at the initial point.
C     In a maximization problem, all uphill moves are accepted and the
C  algorithm continues from that trial point. Downhill moves may be
```

```
C  accepted; the decision is made by the Metropolis criteria. It uses T
C  (temperature) and the size of the downhill move in a probabilistic
C  manner. The smaller T and the size of the downhill move are, the more
C  likely that move will be accepted. If the trial is accepted, the
C  algorithm moves on from that point. If it is rejected, another point
C  is chosen instead for a trial evaluation.
C     Each element of VM periodically adjusted so that half of all
C  function evaluations in that direction are accepted.
C     A fall in T is imposed upon the system with the RT variable by
C  T(i+1) = RT*T(i) where i is the ith iteration. Thus, as T declines,
C  downhill moves are less likely to be accepted and the percentage of
C  rejections rise. Given the scheme for the selection for VM, VM falls.
C  Thus, as T declines, VM falls and SA focuses upon the most promising
C  area for optimization.
C
C  The importance of the parameter T:
C     The parameter T is crucial in using SA successfully. It influences
C  VM, the step length over which the algorithm searches for optima. For
C  a small intial T, the step length may be too small; thus not enough
C  of the function might be evaluated to find the global optima. The user
C  should carefully examine VM in the intermediate output (set IPRINT =
C  1) to make sure that VM is appropriate. The relationship between the
C  initial temperature and the resulting step length is function
C  dependent.
C     To determine the starting temperature that is consistent with
C  optimizing a function, it is worthwhile to run a trial run first. Set
C  RT = 1.5 and T = 1.0. With RT > 1.0, the temperature increases and VM
C  rises as well. Then select the T that produces a large enough VM.
C
C  For modifications to the algorithm and many details on its use,
C  (particularly for econometric applications) see Goffe, Ferrier
C  and Rogers, "Global Optimization of Statistical Functions with
C  the Simulated Annealing," Journal of Econometrics (forthcomming)
C  For a pre-publication copy, contact
C              Bill Goffe
C              Department of Economics
C              Southern Methodist University
C              Dallas, TX  75275
C              h2zr1001 @ smuvm1 (Bitnet)
C              h2zr1001 @ vm.cis.smu.edu (Internet)
C
C  As far as possible, the parameters here have the same name as in
C  the description of the algorithm on pp. 266-8 of Corana et al.
C
C  In this description, SP is single precision, DP is double precision,
C  INT is integer, L is logical and (N) denotes an array of length n.
C  Thus, DP(N) denotes a double precision array of length n.
C
C  Input Parameters:
C    Note: The suggested values generally come from Corana et al. To
C          drastically reduce runtime, see Goffe et al., pp. 17-8 for
C          suggestions on choosing the appropriate RT and NT.
C     N - Number of variables in the function to be optimized. (INT)
C     X - The starting values for the variables of the function to be
C         optimized. (DP(N))
C     MAX - Denotes whether the function should be maximized or
C           minimized. A true value denotes maximization while a false
C           value denotes minimization. Intermediate output (see IPRINT)
C           takes this into account. (L)
```

```
C    RT - The temperature reduction factor. The value suggested by
C         Corana et al. is .85. See Goffe et al. for more advice. (DP)
C    EPS - Error tolerance for termination. If the final function
C          values from the last neps temperatures differ from the
C          corresponding value at the current temperature by less than
C          EPS and the final function value at the current temperature
C          differs from the current optimal function value by less than
C          EPS, execution terminates and IER = 0 is returned. (EP)
C    NS - Number of cycles. After NS*N function evaluations, each
C         element of VM is adjusted so that approximately half of
C         all function evaluations are accepted. The suggested value
C         is 20. (INT)
C    NT - Number of iterations before temperature reduction. After
C         NT*NS*N function evaluations, temperature (T) is changed
C         by the factor RT. Value suggested by Corana et al. is
C         MAX(100, 5*N). See Goffe et al. for further advice. (INT)
C    NEPS - Number of final function values used to decide upon termi-
C           nation. See EPS. Suggested value is 4. (INT)
C    MAXEVL - The maximum number of function evaluations. If it is
C             exceeded, IER = 1. (INT)
C    LB - The lower bound for the allowable solution variables. (DP(N))
C    UB - The upper bound for the allowable solution variables. (DP(N))
C         If the algorithm chooses X(I) .LT. LB(I) or X(I) .GT. UB(I),
C         I = 1, N, a point is from inside is randomly selected. This
C         This focuses the algorithm on the region inside UB and LB.
C         Unless the user wishes to concentrate the search to a par-
C         ticular region, UB and LB should be set to very large positive
C         and negative values, respectively. Note that the starting
C         vector X should be inside this region. Also note that LB and
C         UB are fixed in position, while VM is centered on the last
C         accepted trial set of variables that optimizes the function.
C    C - Vector that controls the step length adjustment. The suggested
C        value for all elements is 2.0. (DP(N))
C    IPRINT - controls printing inside SA. (INT)
C             Values: 0 - Nothing printed.
C                     1 - Function value for the starting value and
C                         summary results before each temperature
C                         reduction. This includes the optimal
C                         function value found so far, the total
C                         number of moves (broken up into uphill,
C                         downhill, accepted and rejected), the
C                         number of out of bounds trials, the
C                         number of new optima found at this
C                         temperature, the current optimal X and
C                         the step length VM. Note that there are
C                         N*NS*NT function evalutations before each
C                         temperature reduction. Finally, notice is
C                         is also given upon achieveing the termination
C                         criteria.
C                     2 - Each new step length (VM), the current optimal
C                         X (XOPT) and the current trial X (X). This
C                         gives the user some idea about how far X
C                         strays from XOPT as well as how VM is adapting
C                         to the function.
C                     3 - Each function evaluation, its acceptance or
C                         rejection and new optima. For many problems,
C                         this option will likely require a small tree
C                         if hard copy is used. This option is best
C                         used to learn about the algorithm. A small
```

```
C                       value for MAXEVL is thus recommended when
C                       using IPRINT = 3.
C            Suggested value: 1
C            Note: For a given value of IPRINT, the lower valued
C                  options (other than 0) are utilized.
C    ISEED1 - The first seed for the random number generator RANMAR.
C             0 .LE. ISEED1 .LE. 31328.
C    ISEED2 - The second seed for the random number generator RANMAR.
C             0 .LE. ISEED2 .LE. 30081. Different values for ISEED1
C             and ISEED2 will lead to an entirely different sequence
C             of trial points and decisions on downhill moves (when
C             maximizing). See Goffe et al. on how this can be used
C             to test the results of SA. (INT)
C
C  Input/Output Parameters:
C    T - On input, the initial temperature. See Goffe et al. for advice.
C        On output, the final temperature. (DP)
C    VM - The step length vector. On input it should encompass the
C         region of interest given the starting value X. For point
C         X(I), the next trial point is selected is from X(I) - VM(I)
C         to  X(I) + VM(I). Since VM is adjusted so that about half
C         of all points are accepted, the input value is not very
C         important (i.e. is the value is off, SA adjusts VM to the
C         correct value). (DP(N))
C
C  Output Parameters:
C    XOPT - The variables that optimize the function. (DP(N))
C    FOPT - The optimal value of the function. (DP)
C    NACC - The number of accepted function evaluations. (INT)
C    NFCNEV - The total number of function evaluations. In a minor
C             point, note that the first evaluation is not used in the
C             core of the algorithm; it simply initializes the
C             algorithm. (INT).
C    NOBDS - The total number of trial function evaluations that
C            would have been out of bounds of LB and UB. Note that
C            a trial point is randomly selected between LB and UB.
C            (INT)
C    IER - The error return number. (INT)
C          Values: 0 - Normal return; termination criteria achieved.
C                  1 - Number of function evaluations (NFCNEV) is
C                      greater than the maximum number (MAXEVL).
C                  2 - The starting value (X) is not inside the
C                      bounds (LB and UB).
C                  99 - Should not be seen; only used internally.
C
C  Work arrays that must be dimensioned in the calling routine:
C       RWK1 (DP(NEPS))  (FSTAR in SA)
C       RWK2 (DP(N))     (XP    "  " )
C       IWK  (INT(N))    (NACP  "  " )
C
C  Required Functions (included):
C    EXPREP - Replaces the function EXP to avoid under- and overflows.
C             It may have to be modified for non IBM-type main-
C             frames. (DP)
C    RMARIN - Initializes the random number generator RANMAR.
C    RANMAR - The actual random number generator. Note that
C             RMARIN must run first (SA does this). It produces uniform
C             random numbers on [0,1]. These routines are from
C             Usenet's comp.lang.fortran, article 2605. For
```

```
C            reference, see "Toward a Universal Random Number
C            Generator" by George Marsaglia and Arif Zaman.
C            Florida State University Report: FSU-SCRI-87-50 (1987).
C            It was later modified by F. James and published in
C            "A Review of Pseudo-random Number Generators." For
C            further information, contact stuart@ads.com. These
C            routines are designed to be portable on any machine
C            with a 24-bit or more mantissa. I have found it produces
C            identical results on a IBM 3081 and a Cray Y-MP.
C
C  Required Subroutines (included):
C    PRTVEC - Prints vectors.
C    PRT0 ... PRT8 - Prints intermediate output.
C    FCN - Function to be optimized. The form is
C             SUBROUTINE FCN(N,X,F)
C             INTEGER N
C             DOUBLE PRECISION  X(N), F
C             ...
C             function code with F = F(X)
C             ...
C             RETURN
C             END
C          Note: This is the same form used in the multivariable
C          minimization algorithms in the IMSL edition 10 library.
C
C  Machine Specific Features:
C    1. EXPREP may have to be modified if used on non-IBM type main-
C       frames. Watch for under- and overflows in EXPREP.
C    2. Some FORMAT statements use G25.18; this may be excessive for
C       some machines.
C    3. RMARIN and RANMAR are designed to be protable; they should not
C       cause any problems.

C  Type all external variables.
      DOUBLE PRECISION  X(*), LB(*), UB(*), C(*), VM(*), FSTAR(*),
     1                  XOPT(*), XP(*), T, EPS, RT, FOPT
      INTEGER  NACP(*), N, NS, NT, NEPS, NACC, MAXEVL, IPRINT,
     1         NOBDS, IER, NFCNEV, ISEED1, ISEED2
      LOGICAL  MAX

C  Type all internal variables.
      DOUBLE PRECISION  F, FP, P, PP, RATIO
      INTEGER  NUP, NDOWN, NREJ, NNEW, LNOBDS, H, I, J, M
      LOGICAL  QUIT

C  Type all functions.
      DOUBLE PRECISION  EXPREP
      REAL  RANMAR

C  Initialize the random number generator RANMAR.
      CALL RMARIN(ISEED1,ISEED2)

C  Set initial values.
      NACC = 0
      NOBDS = 0
      NFCNEV = 0
      IER = 99

      DO 10, I = 1, N
```

```
         XOPT(I) = X(I)
         NACP(I) = 0
10    CONTINUE

      DO 20, I = 1, NEPS
         FSTAR(I) = 1.0D+20
20    CONTINUE

C  If the initial value is out of bounds, notify the user and return
C  to the calling routine.
      DO 30, I = 1, N
         IF ((X(I) .GT. UB(I)) .OR. (X(I) .LT. LB(I))) THEN
            CALL PRT1
            IER = 2
            RETURN
         END IF
30    CONTINUE

C  Evaluate the function with input X and return value as F.
      CALL FCN(N,X,F)

C  If the function is to be minimized, switch the sign of the function.
C  Note that all intermediate and final output switches the sign back
C  to eliminate any possible confusion for the user.
      IF(.NOT. MAX) F = -F
      NFCNEV = NFCNEV + 1
      FOPT = F
      FSTAR(1) = F
      IF(IPRINT .GE. 1) CALL PRT2(MAX,N,X,F)

C  Start the main loop. Note that it terminates if (i) the algorithm
C  succesfully optimizes the function or (ii) there are too many
C  function evaluations (more than MAXEVL).
100   NUP = 0
      NREJ = 0
      NNEW = 0
      NDOWN = 0
      LNOBDS = 0

      DO 400, M = 1, NT
         DO 300, J = 1, NS
            DO 200, H = 1, N

C  Generate XP, the trial value of X. Note use of VM to choose XP.
               DO 110, I = 1, N
                  IF (I .EQ. H) THEN
                     XP(I) = X(I) + (RANMAR()*2.- 1.) * VM(I)
                  ELSE
                     XP(I) = X(I)
                  END IF

C  If XP is out of bounds, select a point in bounds for the trial.
                  IF((XP(I) .LT. LB(I)) .OR. (XP(I) .GT. UB(I))) THEN
                    XP(I) = LB(I) + (UB(I) - LB(I))*RANMAR()
                    LNOBDS = LNOBDS + 1
                    NOBDS = NOBDS + 1
                    IF(IPRINT .GE. 3) CALL PRT3(MAX,N,XP,X,FP,F)
                  END IF
110            CONTINUE
```

```
C  Evaluate the function with the trial point XP and return as FP.
                CALL FCN(N,XP,FP)
                IF(.NOT. MAX) FP = -FP
                NFCNEV = NFCNEV + 1
                IF(IPRINT .GE. 3) CALL PRT4(MAX,N,XP,X,FP,F)

C  If too many function evaluations occur, terminate the algorithm.
                IF(NFCNEV .GE. MAXEVL) THEN
                   CALL PRT5
                   IF (.NOT. MAX) FOPT = -FOPT
                   IER = 1
                   RETURN
                END IF

C  Accept the new point if the function value increases.
                IF(FP .GE. F) THEN
                   IF(IPRINT .GE. 3) THEN
                      WRITE(*,'('' POINT ACCEPTED'')')
                   END IF
                   DO 120, I = 1, N
                      X(I) = XP(I)
120                CONTINUE
                   F = FP
                   NACC = NACC + 1
                   NACP(H) = NACP(H) + 1
                   NUP = NUP + 1

C  If greater than any other point, record as new optimum.
                   IF (FP .GT. FOPT) THEN
                   IF(IPRINT .GE. 3) THEN
                      WRITE(*,'('' NEW OPTIMUM'')')
                   END IF
                      DO 130, I = 1, N
                         XOPT(I) = XP(I)
130                   CONTINUE
                      FOPT = FP
                      NNEW = NNEW + 1
                   END IF

C  If the point is lower, use the Metropolis criteria to decide on
C  acceptance or rejection.
                ELSE
                   P = EXPREP((FP - F)/T)
                   PP = RANMAR()
                   IF (PP .LT. P) THEN
                      IF(IPRINT .GE. 3) CALL PRT6(MAX)
                      DO 140, I = 1, N
                         X(I) = XP(I)
140                   CONTINUE
                      F = FP
                      NACC = NACC + 1
                      NACP(H) = NACP(H) + 1
                      NDOWN = NDOWN + 1
                   ELSE
                      NREJ = NREJ + 1
                      IF(IPRINT .GE. 3) CALL PRT7(MAX)
                   END IF
                END IF
```

```
200         CONTINUE
300      CONTINUE

C  Adjust VM so that approximately half of all evaluations are accepted.
         DO 310, I = 1, N
            RATIO = DFLOAT(NACP(I)) /DFLOAT(NS)
            IF (RATIO .GT. .6) THEN
               VM(I) = VM(I)*(1. + C(I)*(RATIO - .6)/.4)
            ELSE IF (RATIO .LT. .4) THEN
               VM(I) = VM(I)/(1. + C(I)*((.4 - RATIO)/.4))
            END IF
            IF (VM(I) .GT. (UB(I)-LB(I))) THEN
               VM(I) = UB(I) - LB(I)
            END IF
310      CONTINUE

         IF(IPRINT .GE. 2) THEN
            CALL PRT8(N,VM,XOPT,X)
         END IF

         DO 320, I = 1, N
            NACP(I) = 0
320      CONTINUE

400   CONTINUE

      IF(IPRINT .GE. 1) THEN
         CALL PRT9(MAX,N,T,XOPT,VM,FOPT,NUP,NDOWN,NREJ,LNOBDS,NNEW)
      END IF

C  Check termination criteria.
      QUIT = .FALSE.
      FSTAR(1) = F
      IF ((FOPT - FSTAR(1)) .LE. EPS) QUIT = .TRUE.
      DO 410, I = 1, NEPS
         IF (ABS(F - FSTAR(I)) .GT. EPS) QUIT = .FALSE.
410   CONTINUE

C  Terminate SA if appropriate.
      IF (QUIT) THEN
         DO 420, I = 1, N
            X(I) = XOPT(I)
420      CONTINUE
         IER = 0
         IF (.NOT. MAX) FOPT = -FOPT
         IF(IPRINT .GE. 1) CALL PRT10
         RETURN
      END IF

C  If termination criteria is not met, prepare for another loop.
      T = RT*T
      DO 430, I = NEPS, 2, -1
         FSTAR(I) = FSTAR(I-1)
430   CONTINUE
      F = FOPT
      DO 440, I = 1, N
         X(I) = XOPT(I)
440   CONTINUE
```

```
C  Loop again.
      GO TO 100

      END

      FUNCTION  EXPREP(RDUM)
C  This function replaces exp to avoid under- and overflows and is
C  designed for IBM 370 type machines. It may be necessary to modify
C  it for other machines. Note that the maximum and minimum values of
C  EXPREP are such that they has no effect on the algorithm.

      DOUBLE PRECISION  RDUM, EXPREP

      IF (RDUM .GT. 174.) THEN
         EXPREP = 3.69D+75
      ELSE IF (RDUM .LT. -180.) THEN
         EXPREP = 0.0
      ELSE
         EXPREP = EXP(RDUM)
      END IF

      RETURN
      END

      subroutine RMARIN(IJ,KL)
C  This subroutine and the next function generate random numbers. See
C  the comments for SA for more information. The only changes from the
C  orginal code is that (1) the test to make sure that RMARIN runs first
C  was taken out since SA assures that this is done (this test didn't
C  compile under IBM's VS Fortran) and (2) typing ivec as integer was
C  taken out since ivec isn't used. With these exceptions, all following
C  lines are original.

C This is the initialization routine for the random number generator
C     RANMAR()
C NOTE: The seed variables can have values between:    0 <= IJ <= 31328
C                                                      0 <= KL <= 30081
      real U(97), C, CD, CM
      integer I97, J97
      common /raset1/ U, C, CD, CM, I97, J97
      if( IJ .lt. 0  .or.  IJ .gt. 31328  .or.
     *    KL .lt. 0  .or.  KL .gt. 30081 ) then
          print '(A)', ' The first random number seed must have a value
     *between 0 and 31328'
          print '(A)',' The second seed must have a value between 0 and
     *30081'
            stop
      endif
      i = mod(IJ/177, 177) + 2
      j = mod(IJ    , 177) + 2
      k = mod(KL/169, 178) + 1
      l = mod(KL,     169)
      do 2 ii = 1, 97
         s = 0.0
         t = 0.5
         do 3 jj = 1, 24
            m = mod(mod(i*j, 179)*k, 179)
            i = j
```

```
            j = k
            k = m
            l = mod(53*l+1, 169)
            if (mod(l*m, 64) .ge. 32) then
              s = s + t
            endif
            t = 0.5 * t
3         continue
          U(ii) = s
2       continue
        C = 362436.0 / 16777216.0
        CD = 7654321.0 / 16777216.0
        CM = 16777213.0 /16777216.0
        I97 = 97
        J97 = 33
        return
        end

        function ranmar()
        real U(97), C, CD, CM
        integer I97, J97
        common /raset1/ U, C, CD, CM, I97, J97
           uni = U(I97) - U(J97)
           if( uni .lt. 0.0 ) uni = uni + 1.0
           U(I97) = uni
           I97 = I97 - 1
           if(I97 .eq. 0) I97 = 97
           J97 = J97 - 1
           if(J97 .eq. 0) J97 = 97
           C = C - CD
           if( C .lt. 0.0 ) C = C + CM
           uni = uni - C
           if( uni .lt. 0.0 ) uni = uni + 1.0
           RANMAR = uni
        return
        END

        SUBROUTINE PRT1
C  This subroutine prints intermediate output, as does PRT2 through
C  PRT10. Note that if SA is minimizing the function, the sign of the
C  function value and the directions (up/down) are reversed in all
C  output to correspond with the actual function optimization. This
C  correction is because SA was written to maximize functions and
C  it minimizes by maximizing the negative a function.

        WRITE(*,'('' '')')
        WRITE(*,'('' THE STARTING VALUE (X) IS OUTSIDE THE BOUNDS '')')
        WRITE(*,'('' (LB AND UB). EXECUTION TERMINATED WITHOUT ANY'')')
        WRITE(*,'('' OPTIMIZATION. RESPECIFY X, UB OR LB SO THAT  '')')
        WRITE(*,'('' LB(I) .LT. X(I) .LT. UB(I), I = 1, N.        '')')
        WRITE(*,'('' '')')

        RETURN
        END

        SUBROUTINE PRT2(MAX,N,X,F)

        DOUBLE PRECISION  X(*), F
        INTEGER  N
```

```
      LOGICAL  MAX

      WRITE(*,'('' '')')
      CALL PRTVEC(X,N,'INITIAL X')
      IF (MAX) THEN
         WRITE(*,'('' INITIAL F: '',G25.18)') F
      ELSE
         WRITE(*,'('' INITIAL F: '',G25.18)') -F
      END IF
      WRITE(*,'('' '')')

      RETURN
      END

      SUBROUTINE PRT3(MAX,N,XP,X,FP,F)

      DOUBLE PRECISION  XP(*), X(*), FP, F
      INTEGER  N
      LOGICAL  MAX

      WRITE(*,'('' '')')
      CALL PRTVEC(X,N,'CURRENT X')
      IF (MAX) THEN
         WRITE(*,'('' CURRENT F: '',G25.18)') F
      ELSE
         WRITE(*,'('' CURRENT F: '',G25.18)') -F
      END IF
      CALL PRTVEC(XP,N,'TRIAL X')
      WRITE(*,'('' POINT REJECTED SINCE OUT OF BOUNDS'')')

      RETURN
      END

      SUBROUTINE PRT4(MAX,N,XP,X,FP,F)

      DOUBLE PRECISION  XP(*), X(*), FP, F
      INTEGER  N
      LOGICAL  MAX

      WRITE(*,'('' '')')
      CALL PRTVEC(X,N,'CURRENT X')
      IF (MAX) THEN
         WRITE(*,'('' CURRENT F: '',G25.18)') F
         CALL PRTVEC(XP,N,'TRIAL X')
         WRITE(*,'('' RESULTING F: '',G25.18)') FP
      ELSE
         WRITE(*,'('' CURRENT F: '',G25.18)') -F
         CALL PRTVEC(XP,N,'TRIAL X')
         WRITE(*,'('' RESULTING F: '',G25.18)') -FP
      END IF

      RETURN
      END

      SUBROUTINE PRT5

      WRITE(*,'('' '')')
      WRITE(*,'('' TOO MANY FUNCTION EVALUATIONS; CONSIDER '')')
      WRITE(*,'('' INCREASING MAXEVL OR EPS, OR DECREASING '')')
```

```
      WRITE(*,'('' NT OR RT. THESE RESULTS ARE LIKELY TO BE '')')
      WRITE(*,'('' POOR.'')')
      WRITE(*,'('' '')')

      RETURN
      END

      SUBROUTINE PRT6(MAX)

      LOGICAL  MAX

      IF (MAX) THEN
         WRITE(*,'('' THOUGH LOWER, POINT ACCEPTED'')')
      ELSE
         WRITE(*,'('' THOUGH HIGHER, POINT ACCEPTED'')')
      END IF

      RETURN
      END

      SUBROUTINE PRT7(MAX)

      LOGICAL  MAX

      IF (MAX) THEN
         WRITE(*,'('' LOWER POINT REJECTED'')')
      ELSE
         WRITE(*,'('' HIGHER POINT REJECTED'')')
      END IF

      RETURN
      END

      SUBROUTINE PRT8(N,VM,XOPT,X)

      DOUBLE PRECISION  VM(*), XOPT(*), X(*)
      INTEGER  N

      WRITE(*,'('' '')')
      WRITE(*,'('' INTERMEDIATE RESULTS AFTER STEP LENGTH ADJUSTMENT'')'
     1)
      WRITE(*,'('' '')')
      CALL PRTVEC(VM,N,'NEW STEP LENGTH (VM)')
      CALL PRTVEC(XOPT,N,'CURRENT OPTIMAL X')
      CALL PRTVEC(X,N,'CURRENT X')
      WRITE(*,'('' '')')

      RETURN
      END

      SUBROUTINE PRT9(MAX,N,T,XOPT,VM,FOPT,NUP,NDOWN,NREJ,LNOBDS,NNEW)

      DOUBLE PRECISION  XOPT(*), VM(*), T, FOPT
      INTEGER  N, NUP, NDOWN, NREJ, LNOBDS, NNEW, TOTMOV
      LOGICAL  MAX

      TOTMOV = NUP + NDOWN + NREJ

      WRITE(*,'('' '')')
```

```
      WRITE(*,'('' INTERMEDIATE RESULTS BEFORE NEXT TEMPERATURE REDUCTIO
     1N'')')
      WRITE(*,'('' '')')
      WRITE(*,'('' CURRENT TEMPERATURE:            '',G12.5)') T
      IF (MAX) THEN
         WRITE(*,'('' MAX FUNCTION VALUE SO FAR:  '',G25.18)') FOPT
         WRITE(*,'('' TOTAL MOVES:                '',I8)') TOTMOV
         WRITE(*,'(''    UPHILL:                  '',I8)') NUP
         WRITE(*,'(''    ACCEPTED DOWNHILL:       '',I8)') NDOWN
         WRITE(*,'(''    REJECTED DOWNHILL:       '',I8)') NREJ
         WRITE(*,'('' OUT OF BOUNDS TRIALS:       '',I8)') LNOBDS
         WRITE(*,'('' NEW MAXIMA THIS TEMPERATURE:'',I8)') NNEW
      ELSE
         WRITE(*,'('' MIN FUNCTION VALUE SO FAR:  '',G25.18)') -FOPT
         WRITE(*,'('' TOTAL MOVES:                '',I8)') TOTMOV
         WRITE(*,'(''    DOWNHILL:                '',I8)')  NUP
         WRITE(*,'(''    ACCEPTED UPHILL:         '',I8)')  NDOWN
         WRITE(*,'(''    REJECTED UPHILL:         '',I8)')  NREJ
         WRITE(*,'('' TRIALS OUT OF BOUNDS:       '',I8)')  LNOBDS
         WRITE(*,'('' NEW MINIMA THIS TEMPERATURE:'',I8)')  NNEW
      END IF
      CALL PRTVEC(XOPT,N,'CURRENT OPTIMAL X')
      CALL PRTVEC(VM,N,'STEP LENGTH (VM)')
      WRITE(*,'('' '')')

      RETURN
      END

      SUBROUTINE PRT10

      WRITE(*,'('' '')')
      WRITE(*,'('' SA ACHIEVED TERMINATION CRITERIA. IER = 0. '')')
      WRITE(*,'('' '')')

      RETURN
      END

      SUBROUTINE PRTVEC(VECTOR,NCOLS,NAME)
C  This subroutine prints the double precision vector named VECTOR.
C  Elements 1 thru NCOLS will be printed. NAME is a character variable
C  that describes VECTOR. Note that if NAME is given in the call to
C  PRTVEC, it must be enclosed in quotes. If there are more than 10
C  elements in VECTOR, 10 elements will be printed on each line.

      INTEGER NCOLS
      DOUBLE PRECISION VECTOR(NCOLS)
      CHARACTER *(*) NAME

      WRITE(*,1001) NAME

      IF (NCOLS .GT. 10) THEN
         LINES = INT(NCOLS/10.)

         DO 100, I = 1, LINES
            LL = 10*(I - 1)
            WRITE(*,1000) (VECTOR(J),J = 1+LL, 10+LL)
  100    CONTINUE

         WRITE(*,1000) (VECTOR(J),J = 11+LL, NCOLS)
```

```
      ELSE
         WRITE(*,1000) (VECTOR(J),J = 1, NCOLS)
      END IF

1000 FORMAT( 10(G12.5,1X))
1001 FORMAT(/,25X,A)

      RETURN
      END
```

Appendix C

Listing for DF.m

The following program is written in **Mathematica** programming language environment. This program can be used for calculating analytically the value of probability density function of a polynomial of Gaussian random variables at several chosen points.

File AppendixC/DF.m

```
(*  :Title:         DF.m                                                    *)
(*  :Author:        Menner A. Tatang
                    Department of Chemical Engineering
                    Massachusetts Institute of Technology
*)
(*  :Date:          9 February 1994                          *)
(*  :Summary:
     This package is developed for calculating the density function
     of polynomials of independent normal distributed random variables.
*)
(*  :Warning:
     It's better to use numerical value for all coefficients in the
     polynomial chaos expansions for non-linear approximations,
     otherwise the program can not solve the problem.
     ( This program is written to give numerical values.)
*)

BeginPackage["DF`"]

CalDF::usage = "use: CalDF[list,list,list,\"filename\"], for example,
density = CalDF[{1.0 + 0.2 e1 + 0.3 e2, 2.0 + 0.1 e1 + 0.4 e2},
{{z,-5,5,0.5},{w,-5,5,0.5}},{e1,e2},\"filename\"]
gives the values of joint density function for z and w from (z,w) =
(-5,-5) to (5,5) with step 0.5 on both coordinates
and stores them into \"filename\".
This function could also be used for one dimensional problem."

Begin["DF`Private`"]

`DensityN;

CalDF[l1_, l2_, l3_, mfile_] := DensityN[l1,l2,l3,mfile]

DensityN[l1_, l2_, l3_, mfile_] :=
Block[{contvar,nl1,nl2,nl3,sym1,min1,max1,dx1,eqn1,
       sym2,min2,max2,dx2,syml3,sl3,allroots,iroot,
       i1,i2,it1,it2,fn,fni,fns1,fns2,i,j,jmat,jmi,fd,
```

```
        density,di,k,file,fng,x,roots},

(*      specify the lower and upper limits of integration *)

        lowl = -10.0;
        uppl = 10.0;

(*      get the dimensions of the problem and see if they are
        correct
*)

        contvar = True;
        nl1 = Length[l1];
        nl2 = Length[l2];
        nl3 = Length[l3];

        If[nl1 != nl2,
           Print["CalDF Error: Incompatible number of"];
           Print["             variables and domains"];
           contvar = False;
         ];
        If[nl1 > nl3,
           Print["CalDF Error: +/- 1 correlated problem"];
           contvar = False;
         ];

        If[nl1 > 2,
           Print["CalDF Error: More than two dimensions"];
           contvar = False;
         ];

        If[nl3 - nl1 > 1,
           Print["CalDF Error: This version can not do that job"];
           Print["             Modify the program"];
           contvar = False;
         ];

(*      extract the information and see if they are correct *)

        If[contvar,
           sym1 = l2[[1,1]];
           min1 = l2[[1,2]];
           max1 = l2[[1,3]];
           dx1 = l2[[1,4]];
           eqn1 = l1[[1]];
           sym13 = Array[s13,nl3];
           For[i = 1, i <= nl3, i++,
               sym13[[i]] = l3[[i]];
             ];

           If[max1 <= min1,
              Print["CalDF Error: maximum <= minimum problem"];
              contvar = False
            ];
           If[nl1 == 2,
              sym2 = l2[[2,1]];
              min2 = l2[[2,2]];
              max2 = l2[[2,3]];
              dx2 = l2[[2,4]];
```

```
        eqn2 = l1[[2]];
        If[max2 <= min2,
           Print["CalDF Error: maximum <= minimum problem"];
           contvar = False;
          ];
       ];
     ];

    If[contvar,

(*      calculate Jacobian matrix and check its singularity *)

        jmat = Array[jmi,{nl1,nl1}];
        For[i = 1, i <= nl1, i++,
            For[j = 1, j <= nl1, j++,
                jmat[[i,j]] = D[l1[[i]],syml3[[j]]];
               ];
           ];
        fd = Abs[Det[jmat]];
        If[Chop[fd] == 0,
           Print["CalDF Error: Singular covariance problem"];
           contvar = False;
          ];
       ];

    If[contvar,

(*      solve system of equations *)

        If[nl1 == 1,
           allroots = N[Solve[eqn1 == sym1,{syml3[[1]]}]],
           allroots = N[Solve[{eqn1 == sym1,
                               eqn2 == sym2},{syml3[[1]],
                                              syml3[[2]]}]];
          ];
        iroot = Length[allroots];

(*      compute factors for every point *)

(*      for one dimensional problem *)

        i1 = Floor[(max1 - min1)/dx1];
        it1 = i1+1;
        If[min1 + i1 dx1 < max1, it1 = it1 + 1];
        If[nl1 == 1,
           fn = Array[fni,it1];
           fns1 = N[1/Sqrt[2 Pi] Exp[-syml3[[1]]^2/2]/fd] /.
                  allroots;
           For[i = 1, i <= i1+1, i++,
               roots = syml3[[1]] /. allroots
                       /. sym1->N[min1 + (i-1)*dx1];
               fn[[i]] = fns1 /. sym1->N[min1 + (i-1)*dx1];
               For[j = 1, j <= iroot, j++,
                   If[Chop[Im[roots[[j]]]] != 0.0,
                      fn[[i,j]] = 0.0;
                     ];
                  ];
              ];
          If[it1 > i1+1,
```

```
      fn[[it1]] = fns1 /. sym1->N[max1];
      roots = syml3[[1]] /. allroots
              /. sym1->N[max1];
      For[j = 1, j <= iroot, j++,
          If[Chop[Im[roots[[j]]]] != 0.0,
             fn[[it1,j]] = 0.0;
            ];
         ];
    ];

(*          integrate to get the marginal if necessary *)
(*          WARNING: it will slow the computation! *)
(*          you may modify this part for problem *)
(*          nl3 - nl1 > 1, NOTE: multidimensional integration *)
(*          complexity! *)

    If[nl3 - nl1 == 1,
      fng[x_] := 1/Sqrt[2 Pi] Exp[-x^2/2];
      For[i = 1, i <= it1, i++,
          For[j = 1, j <= iroot, j++,
              fn[[i,j]] =
              NIntegrate[fn[[i,j]] fng[x] /.
                         syml3[[2]]->x,{x,lowl,uppl}];
             ];
         ];
    ],

(*          for two dimensional problem *)

    i2 = Floor[(max2 - min2)/dx2];
    it2 = i2+1;
    If[min2 + i2 dx2 < max2, it2 = it2 + 1];
    fn = Array[fni,{it1,it2}];
    fns1 = N[1/Sqrt[2 Pi] Exp[-syml3[[1]]^2/2]/fd] /.
           allroots;
    fns2 = N[1/Sqrt[2 Pi] Exp[-syml3[[2]]^2/2]/fd] /.
           allroots;
    For[i = 1, i <= i1+1, i++,
        For[j = 1, j <= i2+1, j++,
            fn[[i,j]] = fns1*fns2 /.
            {sym1->N[min1 + (i-1)*dx1],
             sym2->N[min2 + (j-1)*dx2]};
            roots = {syml3[[1]],syml3[[2]]} /. allroots
                    /. {sym1->N[min1 + (i-1)*dx1],
                        sym2->N[min2 + (j-1)*dx2]};
            For[k = 1, k <= iroot, k++,
                If[(Chop[Im[roots[[k,1]]]] != 0.0) ||
                   (Chop[Im[roots[[k,2]]]] != 0.0),
                   fn[[i,j,k]] = 0.0;
                  ];
               ];
            ];
      ];
    If[it1 > i1+1,
      For[j = 1, j <= i2+1, j++,
          fn[[it1,j]] = fns1*fns2 /.
           {sym1->N[max1],
            sym2->N[min2 + (j-1)*dx2]};
          roots = {syml3[[1]],syml3[[2]]} /. allroots
```

```
                    /. {sym1->N[max1],
                        sym2->N[min2 + (j-1)*dx2]};
              For[k = 1, k <= iroot, k++,
                  If[(Chop[Im[roots[[k,1]]]] != 0.0) ||
                     (Chop[Im[roots[[k,2]]]] != 0.0),
                     fn[[it1,j,k]] = 0.0;
                    ];
                 ];
             ];
          ];
       If[it2 > i2+1,
          For[i = 1, i <= i1+1, i++,
              fn[[i,it2]] = fns1*fns2 /.
               {sym1->N[min1 + (i-1)*dx1],
                sym2->N[max2]};
              roots = {syml3[[1]],syml3[[2]]} /. allroots
                      /. {sym1->N[min1 + (i-1)*dx1],
                          sym2->N[max2]};
              For[k = 1, k <= iroot, k++,
                  If[(Chop[Im[roots[[k,1]]]] != 0.0) ||
                     (Chop[Im[roots[[k,2]]]] != 0.0),
                     fn[[i,it2,k]] = 0.0;
                    ];
                 ];
             ];
          ];
       If[(it1 > i1+1) && (it2 > i2+1),
          fn[[it1,it2]] = fns1*fns2 /.
          {sym1->N[max1],
           sym2->N[max2]};
           roots = {syml3[[1]],syml3[[2]]} /. allroots
                   /. {sym1->N[max1],
                       sym2->N[max2]};
           For[k = 1, k <= iroot, k++,
               If[(Chop[Im[roots[[k,1]]]] != 0.0) ||
                  (Chop[Im[roots[[k,2]]]] != 0.0),
                  fn[[it1,it2,k]] = 0.0;
                 ];
              ];
          ];
       ];

(*     integrate to get the marginal if necessary *)
(*     WARNING: it will slow the computation! *)
(*     you may modify this part for problem *)
(*     nl3 - nl1 > 1, NOTE: multidimensional integration *)
(*     complexity! *)

       If[nl3 - nl1 == 1,
          fng[x_] := 1/Sqrt[2 Pi] Exp[-x^2/2];
          For[i = 1, i <= it1, i++,
              For[j = 1, j <= it2, j++,
                  For[k = 1, k <= iroot, k++,
                      fn[[i,j,k]] =
                      NIntegrate[fn[[i,j,k]] fng[x] /.
                                 syml3[[3]]->x,
                                 {x,lowl,uppl}];
                     ];
                 ];
```

```
            ];
        ];

(*      collect the density values for one dimensional problem *)

        If[nl1 == 1,
           density = Array[di,{it1,2}];
           For[i = 1, i <= it1, i++,
               density[[i,1]] = N[min1 + (i-1)*dx1];
               density[[i,2]] = 0.0;
               For[j = 1, j <= iroot, j++,
                   density[[i,2]] += Re[fn[[i,j]]];
                  ];
              ];
           If[it1 > i1+1,
               density[[it1,1]] = max1;
            ],

(*         for two dimensional problem *)

           density = Array[di,{it1,it2,3}];
           For[i = 1, i <= it1, i++,
               For[j = 1, j <= it2, j++,
                   density[[i,j,1]] = N[min1 + (i-1)*dx1];
                   density[[i,j,2]] = N[min2 + (j-1)*dx2];
                   density[[i,j,3]] = 0.0;
                   For[k = 1, k <= iroot, k++,
                       density[[i,j,3]] += Re[fn[[i,j,k]]];
                      ];
                  ];
              ];
           If[it1 > i1+1,
              For[j = 1, j <= i2+1, j++,
                  density[[it1,j,1]] = N[max1];
                 ];
            ];
           If[it2 > i2+1,
              For[i = 1, i <= i1+1, i++,
                  density[[i,it2,2]] = N[max2];
                 ];
            ];
           If[(it1 > i1+1) && (it2 > i2+1),
              density[[it1,it2,1]] = N[max1];
              density[[it1,it2,2]] = N[max2];
            ];
         ];

      ];

(*    store and return the result *)

      If[contvar,
         file = OpenWrite[mfile];
         If[nl1 == 2,
            For[i = 1, i <= it1, i++,
                For[j = 1, j <= it2, j++,
                    WriteString[file,
                                FortranForm[density[[i,j,3]]],
                                "\t"];
```

```
                    ];
                WriteString[file,"\n"];
               ],
            For[i = 1, i <= it1, i++,
                WriteString[file,
                            density[[i,1]],"\t",
                            FortranForm[density[[i,2]]],
                            "\n"]
               ];
           ];
         Close[file];
         Return[density],
         Return[0];
        ];
];

End[ ]

Protect[CalDF];
Append[Attributes[CalDF], Locked]

EndPackage[ ]
```

Appendix D

Example of H-Function Approach Result

The following text and figures are stored in the solution file that is written in LaTeX format. This solution file is generated automatically by the H-function program (see reference [158]). The problem description is given in chapter 4.

INPUT FILE

```
# define the distribution of random variables
@Definitions
    tv = gamma,4.0,0.25; # empirical correction factor
    gz = gamma,16.0,0.5; # Graetz number
@End

# define the equation or model
# model from page 302 of "Unit Operations of Chemical Engineering",
# 4th edition, by W.L. McCabe, J.C. Smith, and P. Harriott.
@Equation
    nu = 2.0*tv*gz^0.33333; # Nusselt number empirical correlation
@End
```

RESULT

```
                S T O C H A S T I C   F U N C T I O N   A N A L Y Z E R

TITLE: NUSSELT


*** BEGIN TERM  1 ***
    ADD   CONSTANT     2.00000D+00
    MULT  VARIATE      M= 1,   N= 0,   P= 0,   Q= 1   K= 6.66667D-01   C= 4.00000D+00
          (B,BETA) = ( 3.00000D+00, 1.00000D+00),(
    MULT  VARIATE      M= 1,   N= 0,   P= 0,   Q= 1   K= 9.63480D-13   C= 1.25992D+00
          (B,BETA) = ( 1.56667D+01, 3.33330D-01),(
DISTRIBUTION OF THE TERM (H-FUNCTION PARAMETERS):
          M= 2,   N= 0,   P= 0,   Q= 2   K= 3.21160D-13   C= 2.51984D+00
          (B,BETA) = ( 3.00000D+00, 1.00000D+00),( 1.56667D+01, 3.33330D-01),
                     (
```

MOMENTS OF THE TERM:

3.97186D+00	1.98351D+01	1.19261D+02	8.51450D+02	6.94215D+03
6.38628D+04	6.64116D+05	7.58846D+06	9.48341D+07	1.30579D+09
1.93349D+10	3.07433D+11	5.30136D+12	9.69623D+13	1.88144D+15
3.91688D+16	8.56640D+17	1.97052D+19	4.82547D+20	1.23262D+22

S T O C H A S T I C F U N C T I O N A N A L Y Z E R

MOMENTS OF THE FUNCTION

1	0.39718618D+01	11	0.19334933D+11
2	0.19835137D+02	12	0.30743282D+12
3	0.11926092D+03	13	0.53013622D+13
4	0.85144957D+03	14	0.96962275D+14
5	0.69421466D+04	15	0.18814417D+16
6	0.63862812D+05	16	0.39168776D+17
7	0.66411563D+06	17	0.85663962D+18
8	0.75884600D+07	18	0.19705240D+20
9	0.94834082D+08	19	0.48254659D+21
10	0.13057864D+10	20	0.12326181D+23

S T O C H A S T I C F U N C T I O N A N A L Y Z E R

BEGIN PROGRAM 'TEST'
FOR APPROXIMATING A PROBABILITY DISTRIBUTION FROM ITS MOMENTS
THOMAS W. HILL, 1969

THE RANGE IS (0,INFINITY)

A SOLUTION IS POSSIBLE USING
ONLY THE MOMENTS UP TO 6, BECAUSE
THE VALUE OF THE DETERMINANT
OF ORDER 5 IS -0.98027875D+11

A SOLUTION EXISTS
FOR SPECIFIED MOMENTS

A UNIMODAL DISTRIBUTION EXISTS HAVING
THE SPECIFIED MOMENTS

LAGUERRE APPROXIMATION OF THE PROBABILITY DENSITY FUNCTION

K	K-TH MOMENT	STANDARDIZED MOMENT	K-TH SERIES COEFFICIENT
1	0.39718618D+01	0.39718618D+01	0.98593091D+00
2	0.19835137D+02	0.19835137D+02	0.16699722D+00
3	0.11926092D+03	0.11926092D+03	0.15714748D-02

LAGUERRE APPROXIMATION OF THE PROBABILITY DENSITY FUNCTION

X	P.D.F.	C.D.F.
0.000D+00	0.000000	0.000000
8.000D-02	-0.000251	-0.000010
1.600D-01	0.000367	-0.000005
2.400D-01	0.002017	0.000090
3.200D-01	0.004785	0.000362
4.000D-01	0.008692	0.000901
4.800D-01	0.013711	0.001797
5.600D-01	0.019778	0.003137
6.400D-01	0.026805	0.005000
7.200D-01	0.034683	0.007460
8.000D-01	0.043293	0.010579
8.800D-01	0.052508	0.014411
9.600D-01	0.062202	0.018999
1.040D+00	0.072248	0.024377
1.120D+00	0.082523	0.030568
1.200D+00	0.092913	0.037585
1.280D+00	0.103307	0.045434
1.360D+00	0.113607	0.054111
1.440D+00	0.123721	0.063604
1.520D+00	0.133568	0.073895
1.600D+00	0.143076	0.084961
1.680D+00	0.152184	0.096771
1.760D+00	0.160837	0.109292
1.840D+00	0.168992	0.122485
1.920D+00	0.176612	0.136310
2.000D+00	0.183671	0.150721
2.080D+00	0.190147	0.165674
2.160D+00	0.196025	0.181120
2.240D+00	0.201300	0.197013
2.320D+00	0.205967	0.213304
2.400D+00	0.210030	0.229944
2.480D+00	0.213495	0.246885
2.560D+00	0.216374	0.264080
2.640D+00	0.218682	0.281482
2.720D+00	0.220434	0.299047
2.800D+00	0.221651	0.316730
2.880D+00	0.222354	0.334490
2.960D+00	0.222567	0.352287
3.040D+00	0.222313	0.370082
3.120D+00	0.221618	0.387839
3.200D+00	0.220509	0.405525
3.280D+00	0.219011	0.423105
3.360D+00	0.217152	0.440552
3.440D+00	0.214957	0.457836
3.520D+00	0.212452	0.474933
3.600D+00	0.209665	0.491817

LAGUERRE APPROXIMATION OF THE PROBABILITY DENSITY FUNCTION

X	P.D.F.	C.D.F.
3.680D+00	0.206619	0.508469
3.760D+00	0.203340	0.524867

3.840D+00	0.199851	0.540995
3.920D+00	0.196175	0.556836
4.000D+00	0.192336	0.572376
4.080D+00	0.188353	0.587604
4.160D+00	0.184248	0.602508
4.240D+00	0.180039	0.617079
4.320D+00	0.175744	0.631310
4.400D+00	0.171382	0.645195
4.480D+00	0.166967	0.658729
4.560D+00	0.162516	0.671909
4.640D+00	0.158042	0.684731
4.720D+00	0.153558	0.697195
4.800D+00	0.149077	0.709300
4.880D+00	0.144610	0.721048
4.960D+00	0.140167	0.732439
5.040D+00	0.135757	0.743476
5.120D+00	0.131390	0.754162
5.200D+00	0.127073	0.764500
5.280D+00	0.122813	0.774496
5.360D+00	0.118616	0.784153
5.440D+00	0.114488	0.793477
5.520D+00	0.110435	0.802474
5.600D+00	0.106459	0.811150
5.680D+00	0.102565	0.819511
5.760D+00	0.098757	0.827564
5.840D+00	0.095036	0.835315
5.920D+00	0.091406	0.842773
6.000D+00	0.087867	0.849944
6.080D+00	0.084421	0.856835
6.160D+00	0.081069	0.863455
6.240D+00	0.077812	0.869810
6.320D+00	0.074650	0.875909
6.400D+00	0.071582	0.881758
6.480D+00	0.068610	0.887366
6.560D+00	0.065731	0.892739
6.640D+00	0.062946	0.897886
6.720D+00	0.060253	0.902814
6.800D+00	0.057651	0.907530
6.880D+00	0.055139	0.912042
6.960D+00	0.052716	0.916356
7.040D+00	0.050380	0.920480
7.120D+00	0.048129	0.924420
7.200D+00	0.045961	0.928184
7.280D+00	0.043875	0.931778

LAGUERRE APPROXIMATION OF THE PROBABILITY DENSITY FUNCTION

X	P.D.F.	C.D.F.
7.360D+00	0.041869	0.935207
7.440D+00	0.039941	0.938480
7.520D+00	0.038089	0.941601
7.600D+00	0.036310	0.944577
7.680D+00	0.034604	0.947413
7.760D+00	0.032967	0.950116
7.840D+00	0.031397	0.952691
7.920D+00	0.029893	0.955142

8.000D+00	0.028453	0.957476
8.080D+00	0.027074	0.959697
8.160D+00	0.025754	0.961810
8.240D+00	0.024492	0.963820
8.320D+00	0.023285	0.965731
8.400D+00	0.022132	0.967548
8.480D+00	0.021030	0.969275
8.560D+00	0.019978	0.970915
8.640D+00	0.018973	0.972473
8.720D+00	0.018015	0.973952
8.800D+00	0.017100	0.975357
8.880D+00	0.016228	0.976690
8.960D+00	0.015397	0.977955
9.040D+00	0.014605	0.979155
9.120D+00	0.013851	0.980294
9.200D+00	0.013132	0.981373
9.280D+00	0.012448	0.982396
9.360D+00	0.011797	0.983366
9.440D+00	0.011178	0.984285
9.520D+00	0.010589	0.985156
9.600D+00	0.010029	0.985980
9.680D+00	0.009496	0.986761
9.760D+00	0.008990	0.987501
9.840D+00	0.008510	0.988201
9.920D+00	0.008053	0.988863
1.000D+01	0.007619	0.989490
1.008D+01	0.007208	0.990083
1.016D+01	0.006817	0.990644
1.024D+01	0.006447	0.991175
1.032D+01	0.006095	0.991677
1.040D+01	0.005762	0.992151
1.048D+01	0.005445	0.992599
1.056D+01	0.005146	0.993023
1.064D+01	0.004862	0.993423
1.072D+01	0.004593	0.993801
1.080D+01	0.004338	0.994159
1.088D+01	0.004096	0.994496
1.096D+01	0.003868	0.994814

LAGUERRE APPROXIMATION OF THE PROBABILITY DENSITY FUNCTION

X	P.D.F.	C.D.F.
1.104D+01	0.003651	0.995115
1.112D+01	0.003446	0.995399
1.120D+01	0.003252	0.995667
1.128D+01	0.003069	0.995920
1.136D+01	0.002895	0.996158
1.144D+01	0.002731	0.996384
1.152D+01	0.002576	0.996596
1.160D+01	0.002429	0.996796
1.168D+01	0.002291	0.996985
1.176D+01	0.002160	0.997163
1.184D+01	0.002036	0.997331
1.192D+01	0.001919	0.997489
1.200D+01	0.001808	0.997638
1.208D+01	0.001704	0.997778

1.216D+01	0.001605	0.997911
1.224D+01	0.001512	0.998036
1.232D+01	0.001424	0.998153
1.240D+01	0.001342	0.998264
1.248D+01	0.001263	0.998368
1.256D+01	0.001190	0.998466
1.264D+01	0.001120	0.998558
1.272D+01	0.001054	0.998645
1.280D+01	0.000992	0.998727
1.288D+01	0.000934	0.998804
1.296D+01	0.000879	0.998877
1.304D+01	0.000827	0.998945
1.312D+01	0.000778	0.999009
1.320D+01	0.000732	0.999069
1.328D+01	0.000688	0.999126
1.336D+01	0.000647	0.999180
1.344D+01	0.000609	0.999230
1.352D+01	0.000572	0.999277
1.360D+01	0.000538	0.999322
1.368D+01	0.000506	0.999363
1.376D+01	0.000476	0.999403
1.384D+01	0.000447	0.999440
1.392D+01	0.000420	0.999474
1.400D+01	0.000395	0.999507
1.408D+01	0.000371	0.999537
1.416D+01	0.000348	0.999566
1.424D+01	0.000327	0.999593
1.432D+01	0.000308	0.999619
1.440D+01	0.000289	0.999642
1.448D+01	0.000271	0.999665
1.456D+01	0.000255	0.999686
1.464D+01	0.000239	0.999706

LAGUERRE APPROXIMATION OF THE PROBABILITY DENSITY FUNCTION

X	P.D.F.	C.D.F.
1.472D+01	0.000225	0.999724
1.480D+01	0.000211	0.999742
1.488D+01	0.000198	0.999758
1.496D+01	0.000186	0.999773
1.504D+01	0.000174	0.999788
1.512D+01	0.000164	0.999801
1.520D+01	0.000154	0.999814
1.528D+01	0.000144	0.999826
1.536D+01	0.000135	0.999837
1.544D+01	0.000127	0.999848
1.552D+01	0.000119	0.999857
1.560D+01	0.000112	0.999867
1.568D+01	0.000105	0.999875
1.576D+01	0.000098	0.999883
1.584D+01	0.000092	0.999891
1.592D+01	0.000086	0.999898
1.600D+01	0.000081	0.999905
1.608D+01	0.000076	0.999911

PEARSON CURVES APPROXIMATION

BETA(1) = 0.10128342D+01
BETA(2) = 0.53133006D+01

K = 0.60495892D+00

USE PEARSON TYPE 4
WITH PARAMETERS

R = 0.12469502D+02

M = 0.72347511D+01
A = 0.14697734D+01
V(1) = 0.15430888D+02

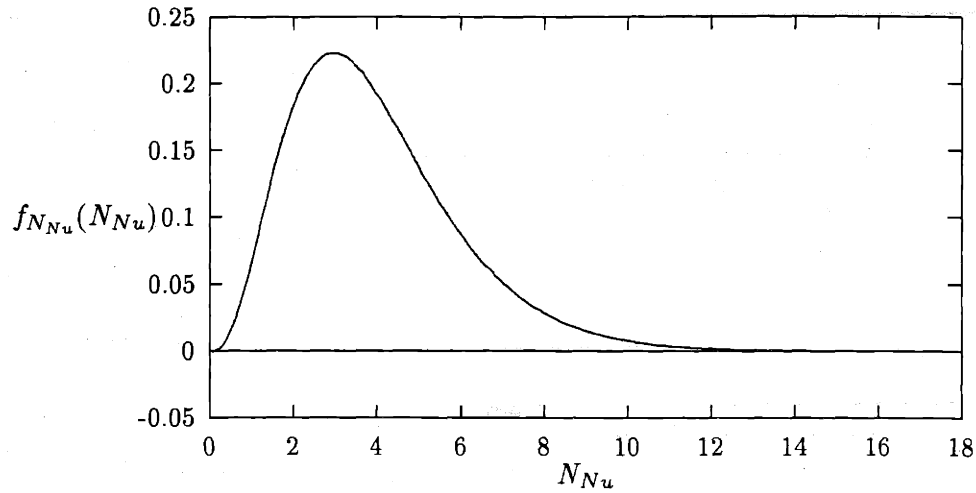


Figure D-1: Probability density function of functional H-variates

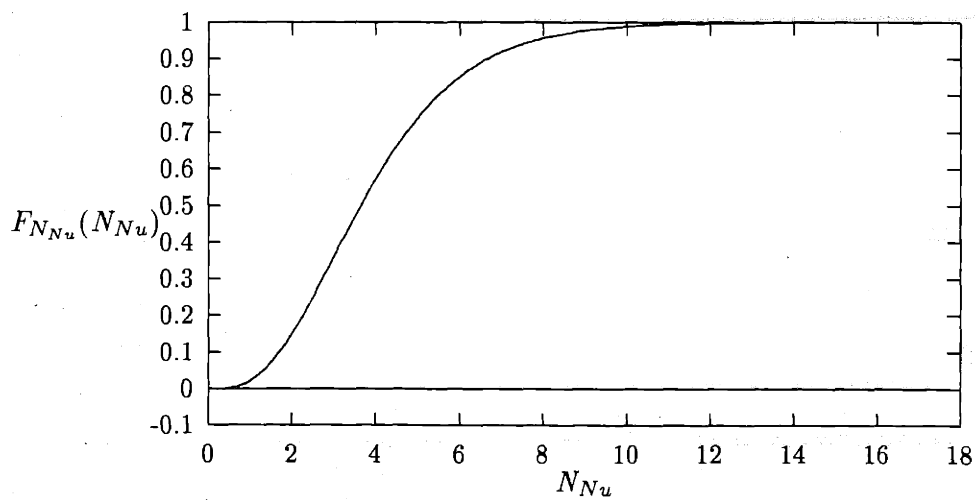


Figure D-2: Cumulative distribution function of functional H-variates

Appendix E

Implementation of the Language of DEMM

E.1 Syntax of the language

The following text describes the syntax rule of the prototype language for DEMM.

```
all:            '\n'
          |     suball

suball:         cVfirst tFormat tNAME
                tProgram program cprogram tEndP

cVfirst:        /* empty */
          |     Vfirst cVfirst

Vfirst:         tVerbatimFirst tVFEnd

cprogram:       /* empty */
          |     program cprogram

program:        cVsecond cdefine cVsecond unit cVsecond

cVsecond:       /* empty */
          |     Vsecond cVsecond

Vsecond:        tVerbatimSecond tVSEnd

cdefine:        /* empty */
          |     define cdefine

define:         sregion
          |     bfregion
          |     pcregion
          |     mvregion
          |     rpregion
          |     rvregion

unit:           tFirst tFILEN
                cdefineF tEnd

cdefineF:       /* empty */
          |     defineF cdefineF
```

```
defineF:        sregionF
          |     bfregionF
          |     pcregionF
          |     mvregionF
          |     rpregionF
          |     rvregionF
          |     mregion

sregion:        tSymbols sstate scstate ';'

sregionF:       tSymbolsF sstate scstate ';'

scstate:        /* empty */
          |     ',' sstate scstate

sstate:         tNAME
          |     tNAMEb
          |     tNAME tEQUAL expr
          |     tNAMEb tEQUAL expr

bfregion:       tSetBF bfstate bfcstate ';'

bfregionF:      tSetFBF bfstate bfcstate ';'

bfcstate:       /* empty */
          |     ',' bfstate bfcstate

bfstate:        tNAME
                '=' '{' bfsymb bfcsymb '}'
          |     tNAMEb
                '=' '{' bfsymb bfcsymb '}'

bfcsymb:        /* empty */
          |     ',' bfsymb bfcsymb

bfsymb:         '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
          |     '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'
          |     tNAME
          |     tNAMEb

pcregion:       tSetPC pcstate pccstate ';'

pcregionF:      tSetFPC pcstate pccstate ';'

pccstate:       /* empty */
          |     ',' pcstate pccstate

pcstate:        tNAME
                '=' '{' pcexpr pccexpr '}'
          |     tNAMEb
                '=' '{' pcexpr pccexpr '}'

pccexpr:        /* empty */
          |     ',' pcexpr pccexpr

pcexpr:         expr

mvregion:       tSetMV mvstate mvcstate ';'
```

```
mvregionF:      tSetFMV mvstate mvcstate ';'

mvcstate:       /* empty */
          |     ',' mvstate mvcstate

mvstate:        tNAME tRELATE related_s
                '=' '{' mvexpr mvcexpr '}'
          |     tNAMEb tRELATE related_s
                '=' '{' mvexpr mvcexpr '}'

related_s:      tNAME
          |     tNAMEb

mvcexpr:        /* empty */
          |     ',' mvexpr mvcexpr

mvexpr:         expr

rpregion:       tSetRP rpstate rpcstate ';'

rpregionF:      tSetFRP rpstate rpcstate ';'

rpcstate:       /* empty */
          |     ',' rpstate rpcstate

rpstate:        tNAME tRELATE related_sp
                '=' '{' rpexpr rpcexpr '}'
          |     tNAMEb tRELATE related_sp
                '=' '{' rpexpr rpcexpr '}'
          |     tNAME tRELATE related_sp
          |     tNAMEb tRELATE related_sp
          |     tNAME tEQUAL expr
          |     tNAMEb tEQUAL expr

related_sp:     tNAME
          |     tNAMEb

rpcexpr:        /* empty */
          |     ',' rpexpr rpcexpr

rpexpr:         '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
          |     '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'
          |     tNAME
          |     tNAMEb

rvregion:       tSetRV rvstate rvcstate ';'

rvregionF:      tSetFRV rvstate rvcstate ';'

rvcstate:       /* empty */
          |     ',' rvstate rvcstate

rvstate:        tNAME tRELATE related_sp
                '=' '{' rvexpr rvcexpr '}'
          |     tNAMEb tRELATE related_sp
                '=' '{' rvexpr rvcexpr '}'
          |     tNAME tRELATE related_sp
          |     tNAMEb tRELATE related_sp
```

```
            |   tNAME tEQUAL expr
            |   tNAMEb tEQUAL expr

rvcexpr:        /* empty */
            |   ',' rvexpr rvcexpr

rvexpr:         '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
            |   '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'
            |   tNAME
            |   tNAMEb

mregion:        tModels mstate mcstate ';'

mcstate:        /* empty */
            |   ',' mstate mcstate

mstate:         tNAME tRELATE related_sp tEQUAL expr
            |   tNAMEb tRELATE related_sp tEQUAL expr
            |   tNAME tEQUAL expr
                '=' '{' mexpr mcexpr '}'
            |   tNAMEb tEQUAL expr
                '=' '{' mexpr mcexpr '}'

mcexpr:         /* empty */
            |   ',' mexpr mcexpr

mexpr:          '{' namexpr namcexpr '}' tRELATE related_sm

related_sm:     '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
            |   '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'

namcexpr:       /* empty */
            |   ',' namexpr namcexpr

namexpr:        tNAME '[' tINT ',' tTO ',' tINT ']'
            |   tNAMEb '[' tINT ',' tTO ',' tINT ']'
            |   tNAME
            |   tNAMEb

expr:           term
            |   expr '+' term
            |   expr '-' term

term:           factor
            |   term '*' factor
            |   term '/' factor

factor:         element
            |   element '^' factor

element:        bracexpr
            |   func
            |   sym
            |   number
            |   '+' element
            |   '-' factor

bracexpr:       '(' expr ')'
```

```
func:          funchead arglist

funchead:      tNAME

arglist:       '(' expr carg ')'

carg:          /* empty */
          |    ',' expr carg

sym:           tNAME
          |    tNAMEb

number:        tINT
          |    tNUMBER
```

E.2 Input file for Flex

The following code is written in **Flex** language environment, and this code is used for generating a scanner for the prototype of DEMM language.

File AppendixE/demmtest.l

```
%{
/*
* a lexer for demm program
*/
#include <stdio.h>
#include <string.h>
#include "new.h"
#include "symtab.h"
#include "demmtest.tab.h"

#undef yywrap()
#define yywrap() (1)

void yyerror(char *);
void initlex();
char *strsave(char *);

extern int lineno;

extern FILE *verbatim_out;

%}

alpha           [a-zA-Z]
digit           [0-9]
equal           ":="
relate          "->"
to              "..."
name            {alpha}("_"|{alpha}|{digit})*
exp             [Ee][-+]?{digit}+
spacel          [\t ]*\n
blank           [\t ]*
int             {digit}+
number          ({digit}+|{digit}+"."{digit}*({exp})?|
                {digit}*"."{digit}+({exp})?|{digit}+{exp})
nameb           {name}("["{blank}({digit}|{name}|",")*{blank}"]")+
```

```
filename        {name}("."{name})*


%start Clear Normal VerbatimFirst Program Symbols
%start VerbatimSecond VerbatimThird
%start First SymbolsF VerbatimFSecond
%start SetBF SetFBF SetPC SetFPC SetMV SetFMV
%start SetRP SetFRP SetRV SetFRV SetModels

%%

<Normal,Program,First>{spacel}    { lineno++; }
<Normal,Program,First>{blank}     ;
<Normal,Program,First>"#".*\n     { lineno++; }
                                  /* ignore comment lines */
<Normal>^"%"{blank}"{"            { BEGIN VerbatimFirst;
                                    return tVerbatimFirst; }
<Normal>^"%"{blank}"%"            { BEGIN Program;
                                    return tProgram; }
<Normal>^"%"[Ff]?(ORMAT|ormat)    { return tFormat; }
<Normal>{name}                    { yylval.c = strsave(yytext);
                                    return tNAME; }
<Normal>.                         { return yytext[0]; }

<VerbatimFirst>{spacel}           { lineno++;
                                    fprintf(verbatim_out,"\n"); }
<VerbatimFirst>^"%"{blank}"}"     { BEGIN Normal; return tVFEnd; }
<VerbatimFirst>.                  { fprintf(verbatim_out,
                                    "%c",yytext[0]); }

<Program>^"%"{blank}"{"                                  { BEGIN
      VerbatimSecond; return tVerbatimSecond; }
<Program>{blank}[Ss](YMBOL|ymbol)[sS]?                   { BEGIN
      Symbols; return tSymbols; }
<Program>{blank}[Ss](ET|et){blank}"["[Bb][Ff]"]"         { BEGIN
      SetBF; return tSetBF; }
<Program>{blank}[Ss](ET|et){blank}"["[Pp][Cc]"]"         { BEGIN
      SetPC; return tSetPC; }
<Program>{blank}[Ss](ET|et){blank}"["[Mm][Vv]"]"         { BEGIN
      SetMV; return tSetMV; }
<Program>{blank}[Ss](ET|et){blank}"["[Rr][Pp]"]"         { BEGIN
      SetRP; return tSetRP; }
<Program>{blank}[Ss](ET|et){blank}"["[Rr][Vv]"]"         { BEGIN
      SetRV; return tSetRV; }
<Program>{blank}[Ss](TART|tart)                          { BEGIN
      First; return tFirst; }
<Program>^"%"{blank}"%"                                  { BEGIN
      VerbatimThird; return tEndP; }
<Program>.                                               { return
      yytext[0]; }

<VerbatimSecond>{spacel}              { lineno++;
      fprintf(verbatim_out,"\n"); }
<VerbatimSecond>^"%"{blank}"}"        { BEGIN Program;
      return tVSEnd; }
<VerbatimSecond>.                     {
      fprintf(verbatim_out,"%c",yytext[0]); }

<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>{spacel}          {
```

```
      lineno++; }
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>{blank}          ;
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>"#".*\n          {
      lineno++; } /* ignore comment lines */
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>";"              { BEGIN
      Program; return yytext[0]; }
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>{nameb}          {
      yylval.c = strsave(yytext); return tNAMEb; }
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>{name}           {
      yylval.c = strsave(yytext); return tNAME; }
<Symbols,SetRP,SetRV>{equal}        { return tEQUAL; }
<SetMV,SetRP,SetRV>{relate}         { return tRELATE; }
<SetBF,SetRP,SetRV>{to}             { return tTO; }
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>{int}            {
      yylval.c = strsave(yytext); return tINT; }
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>{number}         {
      yylval.c = strsave(yytext); return tNUMBER; }
<Symbols,SetBF,SetPC,SetMV,SetRP,SetRV>.                {
      return yytext[0]; }

<First>{blank}[Ss](YMBOL|ymbol)[sS]?                { BEGIN
      SymbolsF; return tSymbolsF; }
<First>{blank}[Ss](ET|et){blank}"["[Bb][Ff]"]"      { BEGIN
      SetFBF; return tSetFBF; }
<First>{blank}[Ss](ET|et){blank}"["[Pp][Cc]"]"      { BEGIN
      SetFPC; return tSetFPC; }
<First>{blank}[Ss](ET|et){blank}"["[Mm][Vv]"]"      { BEGIN
      SetFMV; return tSetFMV; }
<First>{blank}[Ss](ET|et){blank}"["[Rr][Pp]"]"      { BEGIN
      SetFRP; return tSetFRP; }
<First>{blank}[Ss](ET|et){blank}"["[Rr][Vv]"]"      { BEGIN
      SetFRV; return tSetFRV; }
<First>{blank}[Mm](ODEL|odel)[Ss]?                  { BEGIN
      SetModels; return tModels; }
<First>{blank}[Ee](ND|nd)                           { BEGIN
      Program; return tEnd; }
<First>{filename}                                   {
      yylval.c = strsave(yytext); return tFILEN; }
<First>.                                            {
      return yytext[0]; }

<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>{spacel}
     { lineno++; }
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>{blank}   ;
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>"#".*\n
     { lineno++; } /* ignore comment lines */
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>";"
     { BEGIN First; return yytext[0]; }
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>{nameb}
     { yylval.c = strsave(yytext); return tNAMEb; }
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>{name}    {
     yylval.c = strsave(yytext); return tNAME; }
<SymbolsF,SetFRP,SetFRV,SetModels>{equal}           { return
     tEQUAL; }
<SetFMV,SetFRP,SetFRV,SetModels>{relate}            { return
     tRELATE; }
<SetFBF,SetFRP,SetFRV,SetModels>{to}                { return
     tTO; }
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>{int}
```

```
    { yylval.c = strsave(yytext); return tINT; }
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>{number}  {
    yylval.c = strsave(yytext); return tNUMBER; }
<SymbolsF,SetFBF,SetFPC,SetFMV,SetFRP,SetFRV,SetModels>.
    { return yytext[0]; }

<VerbatimThird>{spacel}                    { lineno++;
    fprintf(verbatim_out,"\n"); }
<VerbatimThird>.                           {
    fprintf(verbatim_out,"%c",yytext[0]); }

%%

/* error message */

void yyerror(char *s)
{
   printf("line number %d: %s at %s\n", lineno, s, yytext);
}

/* get it into the proper state before starting */

void initlex()
{
    BEGIN Normal;
}

char *strsave(char *s)
{
    char *t;

    t = (char *)malloc(strlen(s)+1);
    strcpy(t, s);
    return t;
}
```

E.3 Input file for Bison

The following code is written in **Bison** language environment, and this code is used for generating a parser for the prototype of DEMM language.

File AppendixE/demmtest.y

```
%{
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "new.h"
#include "symtab.h"

extern void yyerror(char *);
```

```
short scope = 0;

char *form_name;

FILE *form_file;

short set_element,i,range,start_number;
short range_pc,start_number_pc;
short begin_element,range_total;

bucket *current_set,*current_pc,
       *current_model;
bucket *pc_element,*rp_element,*b;
node *mult_expr;

char *format_file,*form_out,*model_file,
     *number,*pc_element_symbol,*rp_element_content;
char *rp_element_symbol,*m_element;

extern int nsyms,nfuncs,nsbfs,nspcs,nsmvs,nsrps,
           nsrvs,nsmods;
extern int nebfs,nepcs,nemvs,nerps,nervs,nemods,ncmods;

#include "symtab.c"

#include "generateform.c"
#include "check.c"

extern FILE *yyout,*verbatim_out;

%}

%expect 6

%union {
   char *c;
   bucket *b;
   node *n;
   arg  *a;
}

%left '-' '+'
%left '*' '/'
%left UMINUS
%right '^'

%token <c> tFILEN tNAME tNAMEb
%token <c> tINT tNUMBER
%token <c> tEQUAL tRELATE tTO
%token tNormal tFormat tVerbatimFirst tProgram tVerbatimThird
%token tVerbatimSecond tFirst tSymbols
%token tSymbolsF tVFEnd tVSEnd tEnd tEndP
%token tSetBF tSetFBF tSetPC tSetFPC
%token tSetMV tSetFMV tSetRP tSetFRP
%token tSetRV tSetFRV tModels

%type <n> expr term factor element bracexpr func sym number
%type <a> arglist carg
```

```
%type <b> sstate funchead bfsymb pcexpr mvexpr related_s related_sp
%type <b> rpstate rpexpr rvstate rvexpr mstate mexpr namexpr

%%

all:            '\n'
          |     suball

suball:         cVfirst tFormat tNAME
                { if (strcmp($3,"FORTRAN") == MATCH ||
                      strcmp($3,"Fortran") == MATCH ||
                      strcmp($3,"fortran") == MATCH)
                    { format_file = ".f"; }
                  else
                    { if (strcmp($3,"C") == MATCH ||
                          strcmp($3,"c") == MATCH)
                        { format_file = ".c"; }
                      else
                        { if (strcmp($3,"GAMS") == MATCH ||
                              strcmp($3,"Gams") == MATCH ||
                              strcmp($3,"gams") == MATCH)
                            { format_file = ".gms"; }
                          else
                            { if (strcmp($3,"ABACUSS") == MATCH ||
                                  strcmp($3,"Abacuss") == MATCH ||
                                  strcmp($3,"abacuss") == MATCH)
                                { format_file = ".ABACUSS"; }
                              else
                                { yyerror("Format error:
                                  unrecognized format file");
                                  YYERROR;
                                }
                            }
                        }
                    }
                }
                tProgram program cprogram tEndP

cVfirst:        /* empty */
          |     Vfirst cVfirst

Vfirst:         tVerbatimFirst tVFEnd

cprogram:       /* empty */
          |     program cprogram

program:        cVsecond cdefine cVsecond unit cVsecond

cVsecond:       /* empty */
          |     Vsecond cVsecond

Vsecond:        tVerbatimSecond tVSEnd

cdefine:        /* empty */
          |     define cdefine

define:         sregion
          |     bfregion
          |     pcregion
```

```
            |   mvregion
            |   rpregion
            |   rvregion

unit:           tFirst tFILEN
                { scope++;
                  form_name = (char *) malloc(strlen($2)+
                  strlen(".form")+1);
                  model_file = (char *) malloc(strlen($2)+
                  strlen(format_file)+1);
                  strcpy(form_name,$2);
                  strcat(form_name,".form");
                  strcpy(model_file,$2);
                  strcat(model_file,format_file);
                  if (strcmp(format_file,".f") == MATCH)
                    { fprintf(verbatim_out,
                      "      include \'%s\'\n",model_file);
                    }
                  if (strcmp(format_file,".c") == MATCH)
                    { fprintf(verbatim_out,
                      "\n#include \"%s\"\n",model_file);
                    }
                  if (strcmp(format_file,".gms") == MATCH)
                    { fprintf(verbatim_out,
                      "\n$INCLUDE %s\n",model_file);
                    }
                  if (strcmp(format_file,".ABACUSS") == MATCH)
                    { fprintf(verbatim_out,
                      "\n@INCLUDE %s\n",model_file);
                    }
                }
                cdefineF tEnd
                { form_file = fopen(form_name,"w");
                  if (!form_file)
                    { yyerror(
                      "I/O error: could not open form file");
                      YYERROR;
                    }
                  generate_form(form_file);
                  fclose(form_file);
                  form_out = $2;
                  if (check(form_name,form_out,format_file) ==
                      FAIL)
                    { yyerror(
                      "Form error: see result file of form");
                      YYERROR;
                    }
                  free_scope(scope);
                  scope--;
                }

cdefineF:       /* empty */
            |   defineF cdefineF

defineF:        sregionF
            |   bfregionF
            |   pcregionF
            |   mvregionF
            |   rpregionF
```

```
            |   rvregionF
            |   mregion

sregion:        tSymbols sstate scstate ';'

sregionF:       tSymbolsF sstate scstate ';'

scstate:        /* empty */
          |     ',' sstate scstate

sstate:         tNAME
                { $$ = putsym($1,scope,T_SYM,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                }
          |     tNAMEb
                { $$ = putsymb($1,scope,T_SYM,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                }
          |     tNAME tEQUAL expr
                { $$ = putsym($1,scope,T_SYM,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else { $$->expr = $3; }
                }
          |     tNAMEb tEQUAL expr
                { $$ = putsymb($1,scope,T_SYM,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else { $$->expr = $3; }
                }

bfregion:       tSetBF bfstate bfcstate ';'

bfregionF:      tSetFBF bfstate bfcstate ';'

bfcstate:       /* empty */
          |     ',' bfstate bfcstate

bfstate:        tNAME
                { current_set = putsym($1,scope,T_SET_BF,NEW_SYM);
                  if (current_set == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
```

```
                    else
                      { set_element = 0; }
                    }
                    '=' '{' bfsymb bfcsymb '}'
                    { current_set->number_element = set_element;
                      set_element = 0;
                    }
                |   tNAMEb
                    { current_set = putsymb($1,scope,T_SET_BF,NEW_SYM);
                      if (current_set == NULL)
                        { yyerror("Symbols error: try to renew old
                          symbol or access non-existing symbol");
                          YYERROR;
                        }
                      else
                        { set_element = 0; }
                    }
                    '=' '{' bfsymb bfcsymb '}'
                    { current_set->number_element = set_element;
                      set_element = 0;
                    }

bfcsymb:        /* empty */
                |   ',' bfsymb bfcsymb

bfsymb:         '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
                { range = atoi($8) - atoi($4);
                  start_number = atoi($4);
                  for (i = set_element + 1; i <= set_element +
                       range + 1; i++)
                     { $$ = add_element(current_set,i,scope,
                            T_EL_SET_BF,NEW_SYM);
                       if ($$ == NULL)
                         { yyerror("Symbols error: try to renew
                                    old symbol \
                                    or access non-existing
                                    symbol");
                           YYERROR;
                         }
                       else
                         { $$->expr =
                           putnode(NO_NODE,T_SYM,NO_NODE);
                           /* generate the content of this set
                              element */
                           number = (char *) malloc(MAX_LENGTH);
                           sprintf(number,"%d",start_number++);
                           rp_element_content =
                           add_index($2,number);
                           $$->expr->buck = putsymb(
                           rp_element_content,scope,T_SYM,IND_SYM);
                         }
                     }
                  set_element += range + 1;
                }
                |   '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'
                { range = atoi($8) - atoi($4);
                  start_number = atoi($4);
                  for (i = set_element + 1; i <= set_element +
                       range + 1; i++)
```

```
                { $$ = add_element(current_set,i,scope,
                       T_EL_SET_BF,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew
                      old symbol or access
                      non-existing symbol");
                      YYERROR;
                    }
                  else
                    { $$->expr =
                      putnode(NO_NODE,T_SYM,NO_NODE);
                      /* generate the content of this set
                         element */
                      number = (char *) malloc(MAX_LENGTH);
                      sprintf(number,"%d",start_number++);
                      rp_element_content =
                      add_index($2,number);
                      $$->expr->buck = putsymb(
                      rp_element_content,scope,T_SYM,IND_SYM);
                      $$->expr->buck->form_symbol =
                      add_form_index($2,number);
                    }
                }
              set_element += range + 1;
            }
        |   tNAME
            { set_element++;
              $$ = add_element(current_set,
              set_element,scope,T_EL_SET_BF,NEW_SYM);
              if ($$ == NULL)
                { yyerror("Symbols error: try to renew old
                  symbol or access non-existing symbol");
                  YYERROR;
                }
              else
                { $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                  $$->expr->buck =
                  putsym($1,scope,T_SYM,IND_SYM);
                }
            }
        |   tNAMEb
            { set_element++;
              $$ = add_element(current_set,
              set_element,scope,T_EL_SET_BF,NEW_SYM);
              if ($$ == NULL)
                { yyerror("Symbols error: try to renew old
                  symbol or access non-existing symbol");
                  YYERROR;
                }
              else
                { $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                  $$->expr->buck =
                  putsymb($1,scope,T_SYM,IND_SYM);
                }
            }

pcregion:   tSetPC pcstate pccstate ';'
```

```
pcregionF:      tSetFPC pcstate pccstate ';'

pccstate:       /* empty */
          |     ',' pcstate pccstate

pcstate:        tNAME
                { current_set = putsym($1,scope,T_SET_PC,NEW_SYM);
                  if (current_set == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else
                    { set_element = 0; }
                }
                '=' '{' pcexpr pccexpr '}'
                { current_set->number_element = set_element;
                  set_element = 0;
                }
          |     tNAMEb
                { current_set = putsymb($1,scope,T_SET_PC,NEW_SYM);
                  if (current_set == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else
                    { set_element = 0; }
                }
                '=' '{' pcexpr pccexpr '}'
                { current_set->number_element = set_element;
                  set_element = 0;
                }

pccexpr:        /* empty */
          |     ',' pcexpr pccexpr

pcexpr:         expr
                { set_element++;
                  $$ = add_element(current_set,
                  set_element,scope,T_EL_SET_PC,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else
                    { $$->expr = $1; }
                }

mvregion:       tSetMV mvstate mvcstate ';'

mvregionF:      tSetFMV mvstate mvcstate ';'

mvcstate:       /* empty */
          |     ',' mvstate mvcstate

mvstate:        tNAME tRELATE related_s
                { current_set = putsym($1,scope,T_SET_MV,NEW_SYM);
```

```
                    if (current_set == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
                      }
                    else
                      { set_element = 0;
                        $3->related_symbol = current_set->symbol;
                      }
                  }
                  '=' '{' mvexpr mvcexpr '}'
                  { current_set->number_element = set_element;
                    set_element = 0;
                  }
          |       tNAMEb tRELATE related_s
                  { current_set = putsymb($1,scope,T_SET_MV,NEW_SYM);
                    if (current_set == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
                      }
                    else
                      { set_element = 0;
                        $3->related_symbol = current_set->symbol;
                      }
                  }
                  '=' '{' mvexpr mvcexpr '}'
                  { current_set->number_element = set_element;
                    set_element = 0;
                  }

related_s:        tNAME
                  { $$ = putsym($1,scope,T_SET_BF,OLD_SYM);
                    if ($$ == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
                      }
                  }
          |       tNAMEb
                  { $$ = putsymb($1,scope,T_SET_BF,OLD_SYM);
                    if ($$ == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
                      }
                  }

mvcexpr:          /* empty */
          |       ',' mvexpr mvcexpr

mvexpr:           expr
                  { set_element++;
                    $$ = add_element(current_set,
                    set_element,scope,T_EL_SET_MV,NEW_SYM);
                    if ($$ == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
```

```
                    }
                  else
                    { $$->expr = $1;
                    }
                }

rpregion:       tSetRP rpstate rpcstate ';'

rpregionF:      tSetFRP rpstate rpcstate ';'

rpcstate:       /* empty */
          |     ',' rpstate rpcstate

rpstate:        tNAME tRELATE related_sp
                { current_set = putsym($1,scope,T_SET_RP,NEW_SYM);
                  if (current_set == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else
                    { set_element = 0;
                      current_set->related_symbol = $3->symbol;
                    }
                }
                '=' '{' rpexpr rpcexpr '}'
                { current_set->number_element = set_element;
                  if (current_set->number_element >
                      $3->number_element)
                    { yyerror("Set error: unmatch number of
                              elements");
                      YYERROR;
                    }
                  else
                    { current_set->expr = NEW(node);
                      for (i = 1; i <= current_set->number_element;
                           i++)
                         { /* generate expression for this set */
                           number = (char *) malloc(MAX_LENGTH);
                           sprintf(number,"%d",i);
                           pc_element_symbol =
                           add_index($3->symbol,number);
                           pc_element = putsym(pc_element_symbol,
                           scope,T_EL_SET_PC,OLD_SYM);
                           rp_element_symbol = add_index(
                           current_set->symbol,number);
                           rp_element = putsym(rp_element_symbol,
                           scope,T_EL_SET_RP,OLD_SYM);
                           mult_expr = NEW(node);
                           mult_expr = putnode(rp_element->expr,
                           '*',pc_element->expr);
                           current_set->expr= putnode(
                           current_set->expr,'+',mult_expr);
                         }
                      set_element = 0;
                    }
                }
          |     tNAMEb tRELATE related_sp
                { current_set = putsymb($1,scope,T_SET_RP,NEW_SYM);
```

```
    if (current_set == NULL)
      { yyerror("Symbols error: try to renew old
        symbol or access non-existing symbol");
        YYERROR;
      }
    else
      { set_element = 0;
        current_set->related_symbol = $3->symbol;
      }
  }
  '=' '{' rpexpr rpcexpr '}'
  { current_set->number_element = set_element;
    if (current_set->number_element >
        $3->number_element)
      { yyerror("Set error: unmatch number of
                elements");
        YYERROR;
      }
    else
      { current_set->expr = NEW(node);
        for (i = 1; i <= current_set->number_element;
             i++)
           { /* generate expression for this set */
             number = (char *) malloc(MAX_LENGTH);
             sprintf(number,"%d",i);
             pc_element_symbol =
             add_index($3->symbol,number);
             pc_element = putsym(pc_element_symbol,
             scope,T_EL_SET_PC,OLD_SYM);
             rp_element_symbol = add_index(
             current_set->symbol,number);
             rp_element = putsym(rp_element_symbol,
             scope,T_EL_SET_RP,OLD_SYM);
             mult_expr = NEW(node);
             mult_expr = putnode(rp_element->expr,
             '*',pc_element->expr);
             current_set->expr= putnode(
             current_set->expr,'+',mult_expr);
           }
        set_element = 0;
      }
  }
| tNAME tRELATE related_sp
  { current_set = putsym($1,scope,T_SET_RP,NEW_SYM);
    if (current_set == NULL)
      { yyerror("Symbols error: try to renew old
        symbol or access non-existing symbol");
        YYERROR;
      }
    else
      { current_set->related_symbol = $3->symbol;
        current_set->number_element =
        $3->number_element;
        current_set->expr = NEW(node);
        for (i = 1; i <= current_set->number_element;
             i++)
           { $$ = add_element(current_set,i,
             scope,T_EL_SET_RP,NEW_SYM);
             if ($$ == NULL)
```

```
                { yyerror("Symbols error: try to renew
                  old symbol or access
                  non-existing symbol");
                  YYERROR;
                }
              else
                { $$->expr = putnode(NO_NODE,
                  T_SYM,NO_NODE);
                  $$->expr->buck = putsymb($$->symbol,
                  scope,T_SYM,IND_SYM);
                  /* generate expression for this
                     set */
                  number = (char *)
                  malloc(MAX_LENGTH);
                  sprintf(number,"%d",i);
                  pc_element_symbol =
                  add_index($3->symbol,number);
                  pc_element = putsym(
                  pc_element_symbol,scope,
                  T_EL_SET_PC,OLD_SYM);
                  mult_expr = NEW(node);
                  mult_expr = putnode($$->expr,
                  '*',pc_element->expr);
                  current_set->expr= putnode(
                  current_set->expr,'+',mult_expr);
                 }
            }
        }
    }
    tNAMEb tRELATE related_sp
    { current_set = putsymb($1,scope,T_SET_RP,NEW_SYM);
      if (current_set == NULL)
        { yyerror("Symbols error: try to renew old
          symbol or access non-existing symbol");
          YYERROR;
        }
      else
        { current_set->related_symbol = $3->symbol;
          current_set->number_element =
          $3->number_element;
          current_set->expr = NEW(node);
          for (i = 1; i <= current_set->number_element;
               i++)
             { $$ = add_element(current_set,i,
               scope,T_EL_SET_RP,NEW_SYM);
               if ($$ == NULL)
                 { yyerror("Symbols error: try to renew
                   old symbol or access
                   non-existing symbol");
                   YYERROR;
                 }
               else
                 { $$->expr = putnode(NO_NODE,
                   T_SYM,NO_NODE);
                   $$->expr->buck = putsymb($$->symbol,
                   scope,T_SYM,IND_SYM);
                   /* generate expression for this
                      set */
                   number = (char *)
```

```
                        malloc(MAX_LENGTH);
                        sprintf(number,"%d",i);
                        pc_element_symbol = add_index(
                        $3->symbol,number);
                        pc_element = putsym(
                        pc_element_symbol,scope,
                        T_EL_SET_PC,OLD_SYM);
                        mult_expr = NEW(node);
                        mult_expr = putnode($$->expr,
                        '*',pc_element->expr);
                        current_set->expr = putnode(
                        current_set->expr,'+',mult_expr);
                      }
                  }
              }
          }
      |   tNAME tEQUAL expr
          { current_set = putsym($1,scope,T_SET_RP,NEW_SYM);
            if (current_set == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
            else
              { current_set->expr = $3; }
          }
      |   tNAMEb tEQUAL expr
          { current_set = putsymb($1,scope,T_SET_RP,NEW_SYM);
            if (current_set == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
            else
              { current_set->expr = $3; }
          }

related_sp:   tNAME
          { $$ = putsym($1,scope,T_SET_PC,OLD_SYM);
            if ($$ == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
          }
      |   tNAMEb
          { $$ = putsymb($1,scope,T_SET_PC,OLD_SYM);
            if ($$ == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
          }

rpcexpr:      /* empty */
      |   ',' rpexpr rpcexpr

rpexpr:       '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
          { range = atoi($8) - atoi($4);
```

```
   start_number = atoi($4);
   for (i = set_element + 1; i <= set_element +
        range + 1; i++)
      { $$ = add_element(current_set,i,
        scope,T_EL_SET_RP,NEW_SYM);
        if ($$ == NULL)
          { yyerror("Symbols error: try to renew old
            symbol or access non-existing symbol");
            YYERROR;
          }
        else
          { $$->expr =
            putnode(NO_NODE,T_SYM,NO_NODE);
            /* generate the content of this
               set element */
            number = (char *) malloc(MAX_LENGTH);
            sprintf(number,"%d",start_number++);
            rp_element_content =
            add_index($2,number);
            $$->expr->buck = putsymb(
            rp_element_content,scope,T_SYM,IND_SYM);
          }
      }
   set_element += range + 1;
}

|  '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'
{ range = atoi($8) - atoi($4);
   start_number = atoi($4);
   for (i = set_element + 1; i <= set_element +
        range + 1; i++)
      { $$ = add_element(current_set,i,
        scope,T_EL_SET_RP,NEW_SYM);
        if ($$ == NULL)
          { yyerror("Symbols error: try to renew old
            symbol or access non-existing symbol");
            YYERROR;
          }
        else
          { $$->expr =
            putnode(NO_NODE,T_SYM,NO_NODE);
            /* generate the content of this set
               element */
            number = (char *) malloc(MAX_LENGTH);
            sprintf(number,"%d",start_number++);
            rp_element_content =
            add_index($2,number);
            $$->expr->buck = putsymb(
            rp_element_content,scope,T_SYM,IND_SYM);
            $$->expr->buck->form_symbol =
            add_form_index($2,number);
          }
      }
   set_element += range + 1;
}
|  tNAME
{ set_element++;
   $$ = add_element(current_set,
   set_element,scope,T_EL_SET_RP,NEW_SYM);
```

```
                    if ($$ == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
                      }
                    else
                      { $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                        $$->expr->buck =
                        putsym($1,scope,T_SYM,IND_SYM);
                      }
                  }
              |   tNAMEb
                  { set_element++;
                    $$ = add_element(current_set,
                    set_element,scope,T_EL_SET_RP,NEW_SYM);
                    if ($$ == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
                      }
                    else
                      { $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                        $$->expr->buck =
                        putsymb($1,scope,T_SYM,IND_SYM);
                      }
                  }

rvregion:         tSetRV rvstate rvcstate ';'

rvregionF:        tSetFRV rvstate rvcstate ';'

rvcstate:         /* empty */
              |   ',' rvstate rvcstate

rvstate:          tNAME tRELATE related_sp
                  { current_set = putsym($1,scope,T_SET_RV,NEW_SYM);
                    if (current_set == NULL)
                      { yyerror("Symbols error: try to renew old
                        symbol or access non-existing symbol");
                        YYERROR;
                      }
                    else
                      { set_element = 0;
                        current_set->related_symbol = $3->symbol;
                      }
                  }
                  '=' '{' rvexpr rvcexpr '}'
                  { current_set->number_element = set_element;
                    if (current_set->number_element >
                        $3->number_element)
                      { yyerror("Set error: unmatch number of
                        elements");
                        YYERROR;
                      }
                    else
                      { current_set->expr = NEW(node);
                        for (i = 1; i <= current_set->number_element;
                             i++)
                          { /* generate expression for this set */
```

```
                number = (char *) malloc(MAX_LENGTH);
                sprintf(number,"%d",i);
                pc_element_symbol =
                add_index($3->symbol,number);
                pc_element = putsym(pc_element_symbol,
                scope,T_EL_SET_PC,OLD_SYM);
                rp_element_symbol = add_index(
                current_set->symbol,number);
                rp_element = putsym(rp_element_symbol,
                scope,T_EL_SET_RV,OLD_SYM);
                mult_expr = NEW(node);
                mult_expr = putnode(rp_element->expr,
                '*',pc_element->expr);
                current_set->expr= putnode(
                current_set->expr,'+',mult_expr);
              }
          set_element = 0;
        }
    }
|   tNAMEb tRELATE related_sp
    { current_set = putsymb($1,scope,T_SET_RV,NEW_SYM);
      if (current_set == NULL)
        { yyerror("Symbols error: try to renew old
          symbol or access non-existing symbol");
          YYERROR;
        }
      else
        { set_element = 0;
          current_set->related_symbol = $3->symbol;
        }
    }
    '=' '{' rvexpr rvcexpr '}'
    { current_set->number_element = set_element;
      if (current_set->number_element >
          $3->number_element)
        { yyerror("Set error: unmatch number of
          elements");
          YYERROR;
        }
      else
        { current_set->expr = NEW(node);
          for (i = 1; i <= current_set->number_element;
               i++)
             { /* generate expression for this set */
               number = (char *) malloc(MAX_LENGTH);
               sprintf(number,"%d",i);
               pc_element_symbol =
               add_index($3->symbol,number);
               pc_element = putsym(pc_element_symbol,
               scope,T_EL_SET_PC,OLD_SYM);
               rp_element_symbol = add_index(
               current_set->symbol,number);
               rp_element = putsym(rp_element_symbol,
               scope,T_EL_SET_RV,OLD_SYM);
               mult_expr = NEW(node);
               mult_expr = putnode(rp_element->expr,
               '*',pc_element->expr);
               current_set->expr= putnode(
               current_set->expr,'+',mult_expr);
```

```
            }
          set_element = 0;
        }
      }
|     tNAME tRELATE related_sp
      { current_set = putsym($1,scope,T_SET_RV,NEW_SYM);
        if (current_set == NULL)
          { yyerror("Symbols error: try to renew old
            symbol or access non-existing symbol");
            YYERROR;
          }
        else
          { current_set->related_symbol = $3->symbol;
            current_set->number_element =
            $3->number_element;
            current_set->expr = NEW(node);
            for (i = 1; i <= current_set->number_element;
                 i++)
               { $$ = add_element(current_set,i,
                 scope,T_EL_SET_RV,NEW_SYM);
                 if ($$ == NULL)
                   { yyerror("Symbols error: try to renew
                     old symbol or access
                     non-existing symbol");
                     YYERROR;
                   }
                 else
                   { $$->expr =
                     putnode(NO_NODE,T_SYM,NO_NODE);
                     $$->expr->buck = putsymb($$->symbol,
                     scope,T_SYM,IND_SYM);
                     /* generate expression for this
                        set */
                     number = (char *)
                     malloc(MAX_LENGTH);
                     sprintf(number,"%d",i);
                     pc_element_symbol =
                     add_index($3->symbol,number);
                     pc_element = putsym(
                     pc_element_symbol,scope,
                     T_EL_SET_PC,OLD_SYM);
                     mult_expr = NEW(node);
                     mult_expr = putnode($$->expr,
                     '*',pc_element->expr);
                     current_set->expr= putnode(
                     current_set->expr,'+',mult_expr);
                   }
               }
          }
      }
|     tNAMEb tRELATE related_sp
      { current_set = putsymb($1,scope,T_SET_RV,NEW_SYM);
        if (current_set == NULL)
          { yyerror("Symbols error: try to renew old
            symbol or access non-existing symbol");
            YYERROR;
          }
        else
          { current_set->related_symbol = $3->symbol;
```

```
                current_set->number_element =
                $3->number_element;
                current_set->expr = NEW(node);
                for (i = 1; i <= current_set->number_element;
                     i++)
                   { $$ = add_element(current_set,
                     i,scope,T_EL_SET_RV,NEW_SYM);
                     if ($$ == NULL)
                       { yyerror("Symbols error: try to renew
                         old symbol or access
                         non-existing symbol");
                         YYERROR;
                       }
                     else
                       { $$->expr =
                         putnode(NO_NODE,T_SYM,NO_NODE);
                         $$->expr->buck = putsymb($$->symbol,
                         scope,T_SYM,IND_SYM);
                         /* generate expression for this
                            set */
                         number = (char *)
                         malloc(MAX_LENGTH);
                         sprintf(number,"%d",i);
                         pc_element_symbol =
                         add_index($3->symbol,number);
                         pc_element = putsym(
                         pc_element_symbol,scope,
                         T_EL_SET_PC,OLD_SYM);
                         mult_expr = NEW(node);
                         mult_expr = putnode($$->expr,
                         '*',pc_element->expr);
                         current_set->expr = putnode(
                         current_set->expr,'+',mult_expr);
                       }
                   }
              }
          }
    |     tNAME tEQUAL expr
          { current_set = putsym($1,scope,T_SET_RV,NEW_SYM);
            if (current_set == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
            else
              { current_set->expr = $3; }
          }
    |     tNAMEb tEQUAL expr
          { current_set = putsymb($1,scope,T_SET_RV,NEW_SYM);
            if (current_set == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
            else
              { current_set->expr = $3; }
          }

rvcexpr:  /* empty */
```

```
            |   ',' rvexpr rvcexpr

rvexpr:     '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
            { range = atoi($8) - atoi($4);
              start_number = atoi($4);
              for (i = set_element + 1; i <= set_element +
                   range + 1; i++)
                 { $$ = add_element(current_set,
                   i,scope,T_EL_SET_RV,NEW_SYM);
                   if ($$ == NULL)
                     { yyerror("Symbols error: try to renew old
                       symbol or access non-existing symbol");
                       YYERROR;
                     }
                   else
                     { $$->expr =
                       putnode(NO_NODE,T_SYM,NO_NODE);
                       /* generate the content of this set
                          element */
                       number = (char *) malloc(MAX_LENGTH);
                       sprintf(number,"%d",start_number++);
                       rp_element_content =
                       add_index($2,number);
                       $$->expr->buck = putsymb(
                       rp_element_content,scope,T_SYM,IND_SYM);
                     }
                 }
              set_element += range + 1;
            }
            |   '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'
            { range = atoi($8) - atoi($4);
              start_number = atoi($4);
              for (i = set_element + 1; i <= set_element +
                   range + 1; i++)
                 { $$ = add_element(current_set,i,
                   scope,T_EL_SET_RV,NEW_SYM);
                   if ($$ == NULL)
                     { yyerror("Symbols error: try to renew old
                       symbol or access non-existing symbol");
                       YYERROR;
                     }
                   else
                     { $$->expr =
                       putnode(NO_NODE,T_SYM,NO_NODE);
                       /* generate the content of this set
                          element */
                       number = (char *) malloc(MAX_LENGTH);
                       sprintf(number,"%d",start_number++);
                       rp_element_content =
                       add_index($2,number);
                       $$->expr->buck = putsymb(
                       rp_element_content,scope,T_SYM,IND_SYM);
                       $$->expr->buck->form_symbol =
                       add_form_index($2,number);
                     }
                 }
              set_element += range + 1;
            }
            |   tNAME
```

```
                { set_element++;
                  $$ = add_element(current_set,
                  set_element,scope,T_EL_SET_RV,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else
                    { $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                      $$->expr->buck =
                      putsym($1,scope,T_SYM,IND_SYM);
                    }
                }
            |   tNAMEb
                { set_element++;
                  $$ = add_element(current_set,
                  set_element,scope,T_EL_SET_RV,NEW_SYM);
                  if ($$ == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else
                    { $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                      $$->expr->buck =
                      putsymb($1,scope,T_SYM,IND_SYM);
                    }
                }

mregion:        tModels mstate mcstate ';'

mcstate:        /* empty */
            |   ',' mstate mcstate

mstate:         tNAME tRELATE related_sp tEQUAL expr
                { current_model = putsym($1,scope,T_MODEL,NEW_SYM);
                  if (current_model == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                  else
                    { current_model->related_symbol = $3->symbol;
                      current_model->number_element =
                      $3->number_element;
                      current_model->expr = $5;
                      for (i = 1; i <=
                           current_model->number_element; i++)
                         { $$ = add_element(current_model,
                           i,scope,T_EL_MODEL,NEW_SYM);
                           if ($$ == NULL)
                             { yyerror("Symbols error: try to renew
                               old symbol or access
                               non-existing symbol");
                               YYERROR;
                             }
                           else
                             { /* generate the content for this
```

```
                    element set */
                  $$->expr =
                  putnode(NO_NODE,T_SYM,NO_NODE);
                  $$->expr->buck = putsym(
                  $$->symbol,scope,
                  T_CON_MODEL,IND_SYM);
                  $$->expr->buck->form_symbol =
                  $$->form_symbol;
                  /* generate expression for this
                     element set */
                  number = (char *)
                  malloc(MAX_LENGTH);
                  sprintf(number,"%d",i);
                  pc_element_symbol =
                  add_index($3->symbol,number);
                  pc_element = putsym(
                  pc_element_symbol,scope,
                  T_EL_SET_PC,OLD_SYM);
                  mult_expr = NEW(node);
                  mult_expr = putnode(
                  pc_element->expr,'*',
                  current_model->expr);
                  $$->expr->buck->expr = mult_expr;
                }
            }
        }
    }
|   tNAMEb tRELATE related_sp tEQUAL expr
    { current_model =
      putsymb($1,scope,T_MODEL,NEW_SYM);
      if (current_model == NULL)
        { yyerror("Symbols error: try to renew old
          symbol or access non-existing symbol");
          YYERROR;
        }
      else
        { current_model->related_symbol = $3->symbol;
          current_model->number_element =
          $3->number_element;
          current_model->expr = $5;
          for (i = 1; i <=
               current_model->number_element; i++)
             { $$ = add_element(current_model,
               i,scope,T_EL_MODEL,NEW_SYM);
               if ($$ == NULL)
                 { yyerror("Symbols error: try to renew
                   old symbol or access
                   non-existing symbol");
                   YYERROR;
                 }
               else
                 { /* generate the content for this
                      element set */
                   $$->expr =
                   putnode(NO_NODE,T_SYM,NO_NODE);
                   $$->expr->buck = putsym($$->symbol,
                   scope,T_CON_MODEL,IND_SYM);
                   $$->expr->buck->form_symbol =
                   $$->form_symbol;
```

```
                        /* generate expression for this
                           element set */
                        number = (char *)
                        malloc(MAX_LENGTH);
                        sprintf(number,"%d",i);
                        pc_element_symbol =
                        add_index($3->symbol,number);
                        pc_element = putsym(
                        pc_element_symbol,scope,
                        T_EL_SET_PC,OLD_SYM);
                        mult_expr = NEW(node);
                        mult_expr = putnode(
                        pc_element->expr,'*',
                        current_model->expr);
                        $$->expr->buck->expr = mult_expr;
                      }
                  }
              }
          }
    |     tNAME tEQUAL expr
          { current_model = putsym($1,scope,T_MODEL,NEW_SYM);
            if (current_model == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
            else
              { current_model->expr = $3;
                set_element = 0;
              }
          }
          '=' '{' mexpr mcexpr '}'
          { current_model->number_element = set_element;
            set_element = 0;
          }
    |     tNAMEb tEQUAL expr
          { current_model =
            putsymb($1,scope,T_MODEL,NEW_SYM);
            if (current_model == NULL)
              { yyerror("Symbols error: try to renew old
                symbol or access non-existing symbol");
                YYERROR;
              }
            else
              { current_model->expr = $3;
                set_element = 0;
              }
          }
          '=' '{' mexpr mcexpr '}'
          { current_model->number_element = set_element;
            set_element = 0;
          }

mcexpr:   /* empty */
    |     ',' mexpr mcexpr

mexpr:    '{'
          { range_total = 0; }
          namexpr namcexpr '}' tRELATE related_sm
```

```
                { if (range_total > range_pc + 1)
                    { yyerror("Set error: unmatch number of
                              elements");
                      YYERROR;
                    }
                  begin_element = set_element - range_total;
                  for (i = begin_element + 1; i <= set_element;
                       i++)
                     { /* get the element */
                       number = (char *) malloc(MAX_LENGTH);
                       sprintf(number,"%d",i);
                       m_element =
                       add_index(current_model->symbol,number);
                       $$ = putsym(m_element,scope,
                       T_EL_MODEL,OLD_SYM);
                       /* generate expression for this element
                          set */
                       number = (char *) malloc(MAX_LENGTH);
                       sprintf(number,"%d",start_number_pc);
                       pc_element_symbol = add_index(
                       current_pc->symbol,number);
                       pc_element = putsym(pc_element_symbol,
                       scope,T_EL_SET_PC,OLD_SYM);
                       mult_expr = NEW(node);
                       mult_expr = putnode(pc_element->expr,
                       '*',current_model->expr);
                       $$->expr->buck->expr = mult_expr;
                       start_number_pc++;
                     }
                }

related_sm:     '{' tNAME '[' tINT ',' tTO ',' tINT ']' '}'
                { range_pc = atoi($8) - atoi($4);
                  start_number_pc = atoi($4);
                  current_pc = putsym($2,scope,T_SET_PC,OLD_SYM);
                }
          |     '{' tNAMEb '[' tINT ',' tTO ',' tINT ']' '}'
                { range_pc = atoi($8) - atoi($4);
                  start_number_pc = atoi($4);
                  current_pc = putsym($2,scope,T_SET_PC,OLD_SYM);
                }

namcexpr:       /* empty */
          |     ',' namexpr namcexpr

namexpr:        tNAME '[' tINT ',' tTO ',' tINT ']'
                { range = atoi($7) - atoi($3);
                  start_number = atoi($3);
                  for (i = set_element + 1; i <= set_element +
                       range + 1; i++)
                     { $$ = add_element(current_model,
                       i,scope,T_EL_MODEL,NEW_SYM);
                       if ($$ == NULL)
                         { yyerror("Symbols error: try to renew old
                           symbol or access non-existing symbol");
                           YYERROR;
                         }
                       else
                         { /* generate the content of this element
```

```
                    set */
                  number = (char *) malloc(MAX_LENGTH);
                  sprintf(number,"%d",start_number++);
                  rp_element_content =
                  add_index($1,number);
                  $$->expr = putnode(NO_NODE,
                  T_SYM,NO_NODE);
                  $$->expr->buck = putsymb(
                  rp_element_content,scope,
                  T_CON_MODEL,IND_SYM);
                }
              }
        set_element += range + 1;
        range_total += range + 1;
      }
    |  tNAMEb '[' tINT ',' tTO ',' tINT ']'
      { range = atoi($7) - atoi($3);
        start_number = atoi($3);
        for (i = set_element + 1; i <= set_element +
             range + 1; i++)
           { $$ = add_element(current_model,
             i,scope,T_EL_MODEL,NEW_SYM);
             if ($$ == NULL)
               { yyerror("Symbols error: try to renew old
                 symbol or access non-existing symbol");
                 YYERROR;
               }
             else
               { /* generate the content of this element
                    set */
                 number = (char *) malloc(MAX_LENGTH);
                 sprintf(number,"%d",start_number++);
                 rp_element_content =
                 add_index($1,number);
                 $$->expr =
                 putnode(NO_NODE,T_SYM,NO_NODE);
                 $$->expr->buck = putsymb(
                 rp_element_content,scope,
                 T_CON_MODEL,IND_SYM);
                 $$->expr->buck->form_symbol =
                 add_form_index($1,number);
               }
           }
        set_element += range + 1;
        range_total += range + 1;
      }
    |  tNAME
      { set_element += 1;
        range_total += 1;
        $$ = add_element(current_model,
        set_element,scope,T_EL_MODEL,NEW_SYM);
        if ($$ == NULL)
          { yyerror("Symbols error: try to renew old
            symbol or access non-existing symbol");
            YYERROR;
          }
        else
          { /* generate the content of this element
               set */
```

```
                $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                $$->expr->buck =
                putsym($1,scope,T_CON_MODEL,IND_SYM);
              }
            }
        |   tNAMEb
            { set_element += 1;
              range_total += 1;
              $$ = add_element(current_model,
              set_element,scope,T_EL_MODEL,NEW_SYM);
              if ($$ == NULL)
                { yyerror("Symbols error: try to renew old
                  symbol or access non-existing symbol");
                  YYERROR;
                }
              else
                { /* generate the content of this element
                     set */
                  $$->expr = putnode(NO_NODE,T_SYM,NO_NODE);
                  $$->expr->buck =
                  putsymb($1,scope,T_CON_MODEL,IND_SYM);
                }
            }

expr:       term { $$ = $1; }
        |   expr '+' term { $$ = putnode($1,'+',$3); }
        |   expr '-' term { $$ = putnode($1,'-',$3); }

term:       factor { $$ = $1; }
        |   term '*' factor { $$ = putnode($1,'*',$3); }
        |   term '/' factor { $$ = putnode($1,'/',$3); }

factor:     element { $$ = $1; }
        |   element '^' factor { $$ = putnode($1,'^',$3); }

element:    bracexpr { $$ = $1; }
        |   func     { $$ = $1; }
        |   sym      { $$ = $1; }
        |   number   { $$ = $1; }
        |   '+' element %prec UMINUS
            { $$ = $2; }
        |   '-' factor %prec UMINUS
            { $$ =
              putnode(putnode(NO_NODE,T_NUM,NO_NODE),'-',$2); }

bracexpr:   '(' expr ')' { $$ = $2; }

func:       funchead arglist
            { $$ = putnode(NO_NODE,T_FUNC,NO_NODE);
              $$->buck = $1;
              $$->argument = $2;
            }

funchead:   tNAME
            { $$ = putsymexpr($1,scope,T_FUNC,IND_SYM);
              if ($$ == NULL)
                { yyerror("Symbols error: try to renew old
                  symbol or access non-existing symbol");
                  YYERROR;
```

```
                    }
                }

arglist:        '(' expr carg ')'
                { $$ = putarg($2);
                  $$->argnext = $3;
                }

carg:           /* empty */ { $$ = NO_ARG; }
          |     ',' expr carg
                { $$ = putarg($2);
                  $$->argnext = $3;
                }

sym:            tNAME
                { $$ = putnode(NO_NODE,T_SYM,NO_NODE);
                  $$->buck = putsymexpr($1,scope,T_SYM,IND_SYM);
                  if ($$->buck == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                }
          |     tNAMEb
                { $$ = putnode(NO_NODE,T_SYM,NO_NODE);
                  $$->buck = putsymbexpr($1,scope,T_SYM,IND_SYM);
                  if ($$->buck == NULL)
                    { yyerror("Symbols error: try to renew old
                      symbol or access non-existing symbol");
                      YYERROR;
                    }
                }

number:         tINT
                { $$ = putnode(NO_NODE,T_NUM,NO_NODE);
                  $$->value = atoi($1);
                  $$->form_value = $1;
                }
          |     tNUMBER
                { $$ = putnode(NO_NODE,T_NUM,NO_NODE);
                  $$->buck = putsymbexpr($1,scope,T_SYM,IND_SYM);
                  $$->form_value = transform_number($1);
                  $$->buck->form_symbol = $$->form_value;
                }

%%
```

E.4 Symbol table routine

The following code is used for defining the symbol table for the prototype of DEMM language.

File AppendixE/symtab.c

```
bucket **symtab;
bucket *firstsymbol;
bucket *lastsymbol;

int hash(char *key)
{
  register char *cp;
  register int k;

  cp = key;
  k = 0;
  while (*cp)
    k = ((k << 1) ^ (*cp++)) & 0x3fff;

  return (k % TABSIZE);
}

char *copys(char *s)
{
  register int i;
  register char *cp;
  register char *result;

  i = 1;
  for (cp = s; *cp; cp++)
    i++;

  result =  malloc((unsigned int)i+1);
  strcpy(result, s);
  return (result);
}

void tabinit()
{
/*   register int i; JF unused */

  symtab = NEW2(TABSIZE, bucket *);

  firstsymbol = NULL;
  lastsymbol = NULL;
}

char **get_name_and_order(char *s)
{
     int  i,j,start,second;
     char *name,*order;
     char **result;

     name = (char *) malloc(strlen(s)+1);
     order = (char *) malloc(strlen(s)+1);
     start = 0;
     second = 1;
     for (i = 0; i <= strlen(s); i++)
        { if (s[i] == '[') { start = 1; second = 0; j = i+1; }
          if (s[i] == ']')   second = 1;
          if (s[i] != '[' && start == 0)
              name[i] = s[i];
          if (s[i] != ']' && second == 0)
              order[i-j] = s[i];
```

```
        }
      name[j-1] = '\0';
      order[strlen(s)-j] = '\0';
      result = (char **) malloc(2*strlen(s)+1);
      result[0] = name;
      result[1] = order;
      return (result);
}

char *transform(char *s)
{
      int  i,j;
      char *result;

      result = (char *) malloc(MAX_LENGTH);
      strcpy(result,"[");
      j = 1;
      for (i = 0; i <= strlen(s); i++)
         { if (s[i] == '[')
             { strcat(result,"$<");
               j += 2;
             }
           else
             { if (s[i] == ']')
                 { strcat(result,"$>");
                   j += 2;
                 }
               else
                 { result[j] = s[i];
                   j += 1;
                 }
             }
         }
      strcat(result,"]");
      return (result);
}

char *transform_number(char *s)
{
      char *result;

      result = (char *) malloc(strlen(s)+8);
      strcpy(result,"[");
      strcat(result,s);
      strcat(result,"]");
      return(result);
}


bucket *putsym(char *key, short scope,short class,short option)
{
  register int hashval;
  register bucket *bp;
  register int found;
  register short pscope;
  register short pclass;
  register short poption;

  hashval = hash(key);  /* compute the tag number */
```

```
    bp = symtab[hashval]; /* point to that tag */
    pclass = class;       /* copy class and */
    poption = option;     /* option parameters */

    found = NOT_FIND;
    pscope = MATCH;  /* assume there is a symbol with scope */

    if (option == NEW_SYM)
      {
        while (bp != NULL && found == NOT_FIND && pscope == MATCH)
          {
            if (bp->scope == scope)
              {
                if (strcmp(key, bp->symbol) == MATCH)
                    found = FOUND;
                else
                    bp = bp->link;
              }
            else pscope = NOT_MATCH;  /* only different scope */
          }
      }
    else
      {
        while (bp != NULL && found == NOT_FIND)
          {
            if (strcmp(key, bp->symbol) == MATCH && bp->class == pclass)
                found = FOUND;
            else
                bp = bp->link;
          }
      }

    if (found == FOUND && poption == OLD_SYM)
      {
        if (bp->class == pclass)
          { return (bp); }
        else
          { return NULL; }
      }

    /* try to renew old symbol, give error message */

    if (found == FOUND && poption == NEW_SYM)
      {
        if (bp->class == pclass)
          { return NULL; }
        else
          { found = NOT_FIND; }
      }

    /* try to access non-existing symbol, give error message */

    if (found == NOT_FIND && poption == OLD_SYM)
      {
        return NULL;
      }

    /* got the symbol, then return it */
```

```
  if (found == FOUND && poption == IND_SYM)
    {
      if (bp->class == pclass)
        { return (bp); }
      else
        { found = NOT_FIND; }
    }

  /* create the symbol, then return it */

  if (found == NOT_FIND && poption <= IND_SYM)
    {
      switch(pclass)
        {
          case T_SYM: nsyms++; break;
          case T_FUNC: nfuncs++; break;
          case T_SET_BF: nsbfs++; break;
          case T_SET_PC: nspcs++; break;
          case T_SET_MV: nsmvs++; break;
          case T_SET_RP: nsrps++; break;
          case T_SET_RV: nsrvs++; break;
          case T_MODEL: nsmods++; break;
          case T_EL_SET_BF: nebfs++; break;
          case T_EL_SET_PC: nepcs++; break;
          case T_EL_SET_MV: nemvs++; break;
          case T_EL_SET_RP: nerps++; break;
          case T_EL_SET_RV: nervs++; break;
          case T_EL_MODEL: nemods++; break;
          case T_CON_MODEL: ncmods++; break;
        }

      bp = NEW(bucket);
      bp->link = symtab[hashval];
      bp->next = NULL;
      bp->symbol = copys(key);
      bp->form_symbol = copys(key);
      bp->class = pclass;
      bp->scope = scope;
      bp->expr = NULL;

      if (firstsymbol == NULL)
        {
          firstsymbol = bp;
          lastsymbol = bp;
        }
      else
        {
          lastsymbol->next = bp;
          lastsymbol = bp;
        }

      symtab[hashval] = bp;
      return (bp);
    }

}

bucket *putsymexpr(char *key, short scope,short class,short option)
{
```

```
register int hashval;
register bucket *bp;
register int found;
register short pscope;
register short pclass;
register short poption;

hashval = hash(key);  /* compute the tag number */
bp = symtab[hashval]; /* point to that tag */
pclass = class;       /* copy class and */
poption = option;     /* option parameters */

found = NOT_FIND;
pscope = MATCH;  /* assume there is a symbol with scope */

if (option == NEW_SYM)
  {
    while (bp != NULL && found == NOT_FIND && pscope == MATCH)
      {
        if (bp->scope == scope)
          {
            if (strcmp(key, bp->symbol) == MATCH)
                found = FOUND;
            else
                bp = bp->link;
          }
        else pscope = NOT_MATCH;  /* only different scope */
      }
  }
else
  {
    while (bp != NULL && found == NOT_FIND)
      {
        if (strcmp(key, bp->symbol) == MATCH)
            found = FOUND;
        else
            bp = bp->link;
      }
  }

/* try to renew old symbol, give error message */

if (found == FOUND && poption == NEW_SYM)
  {
    return NULL;
  }

/* try to access non-existing symbol, give error message */

if (found == NOT_FIND && poption == OLD_SYM)
  {
    return NULL;
  }

/* got the symbol, then return it */

if (found == FOUND && poption >= IND_SYM)
  {
    return (bp);
```

```
    }

  /* create the symbol, then return it */

  if (found == NOT_FIND && poption <= IND_SYM)
    {
      switch(pclass)
       {
          case T_SYM: nsyms++; break;
          case T_FUNC: nfuncs++; break;
          case T_SET_BF: nsbfs++; break;
          case T_SET_PC: nspcs++; break;
          case T_SET_MV: nsmvs++; break;
          case T_SET_RP: nsrps++; break;
          case T_SET_RV: nsrvs++; break;
          case T_MODEL: nsmods++; break;
          case T_EL_SET_BF: nebfs++; break;
          case T_EL_SET_PC: nepcs++; break;
          case T_EL_SET_MV: nemvs++; break;
          case T_EL_SET_RP: nerps++; break;
          case T_EL_SET_RV: nervs++; break;
          case T_EL_MODEL: nemods++; break;
          case T_CON_MODEL: ncmods++; break;
       }

      bp = NEW(bucket);
      bp->link = symtab[hashval];
      bp->next = NULL;
      bp->symbol = copys(key);
      bp->form_symbol = copys(key);
      bp->class = pclass;
      bp->scope = scope;
      bp->expr = NULL;

      if (firstsymbol == NULL)
        {
          firstsymbol = bp;
          lastsymbol = bp;
        }
      else
        {
          lastsymbol->next = bp;
          lastsymbol = bp;
        }

      symtab[hashval] = bp;
      return (bp);
    }

}


bucket *putsymb(char *key, short scope,short class,short option)
{
  register int hashval;
  register bucket *bp;
  register int found;
  register short pscope;
  register short pclass;
```

```
register short poption;

hashval = hash(key);  /* compute the tag number */
bp = symtab[hashval]; /* point to that tag */
pclass = class;       /* copy class and */
poption = option;     /* option parameters */

found = NOT_FIND;
pscope = MATCH;  /* assume there is a symbol with scope */

if (option == NEW_SYM)
  {
    while (bp != NULL && found == NOT_FIND && pscope == MATCH)
      {
        if (bp->scope == scope)
          {
            if (strcmp(key, bp->symbol) == MATCH)
                found = FOUND;
            else
                bp = bp->link;
          }
        else pscope = NOT_MATCH;  /* only different scope */
      }
  }
else
  {
    while (bp != NULL && found == NOT_FIND)
      {
        if (strcmp(key, bp->symbol) == MATCH && bp->class == pclass)
            found = FOUND;
        else
            bp = bp->link;
      }
  }

if (found == FOUND && poption == OLD_SYM)
  {
    if (bp->class == pclass)
      { return (bp); }
    else
      { return NULL; }
  }

/* try to renew old symbol, give error message */

if (found == FOUND && poption == NEW_SYM)
  {
    if (bp->class == pclass)
      { return NULL; }
    else
      { found = NOT_FIND; }
  }

/* try to access non-existing symbol, give error message */

if (found == NOT_FIND && poption == OLD_SYM)
  {
    return NULL;
  }
```

```
/* got the symbol, then return it if in the same class */
/* otherwise give error message */

if (found == FOUND && poption == IND_SYM)
  {
    if (bp->class == pclass)
      { return (bp); }
    else
      { found = NOT_FIND; }
  }

/* create the symbol, then return it */

if (found == NOT_FIND && poption <= IND_SYM)
  {
    switch(pclass)
      {
        case T_SYM: nsyms++; break;
        case T_FUNC: nfuncs++; break;
        case T_SET_BF: nsbfs++; break;
        case T_SET_PC: nspcs++; break;
        case T_SET_MV: nsmvs++; break;
        case T_SET_RP: nsrps++; break;
        case T_SET_RV: nsrvs++; break;
        case T_MODEL: nsmods++; break;
        case T_EL_SET_BF: nebfs++; break;
        case T_EL_SET_PC: nepcs++; break;
        case T_EL_SET_MV: nemvs++; break;
        case T_EL_SET_RP: nerps++; break;
        case T_EL_SET_RV: nervs++; break;
        case T_EL_MODEL: nemods++; break;
        case T_CON_MODEL: ncmods++; break;
      }

    bp = NEW(bucket);
    bp->link = symtab[hashval];
    bp->next = NULL;
    bp->symbol = copys(key);
    bp->form_symbol = transform(key);
    bp->class = pclass;
    bp->scope = scope;
    bp->expr = NULL;

    if (firstsymbol == NULL)
      {
        firstsymbol = bp;
        lastsymbol = bp;
      }
    else
      {
        lastsymbol->next = bp;
        lastsymbol = bp;
      }

    symtab[hashval] = bp;
    return (bp);
  }
```

```
}

bucket *putsymbexpr(char *key, short scope,short class,short option)
{
  register int hashval;
  register bucket *bp;
  register int found;
  register short pscope;
  register short pclass;
  register short poption;

  hashval = hash(key);  /* compute the tag number */
  bp = symtab[hashval]; /* point to that tag */
  pclass = class;       /* copy class and */
  poption = option;     /* option parameters */

  found = NOT_FIND;
  pscope = MATCH;  /* assume there is a symbol with scope */

  if (option == NEW_SYM)
    {
      while (bp != NULL && found == NOT_FIND && pscope == MATCH)
        {
          if (bp->scope == scope)
            {
              if (strcmp(key, bp->symbol) == MATCH)
                  found = FOUND;
              else
                  bp = bp->link;
            }
          else pscope = NOT_MATCH;  /* only different scope */
        }
    }
  else
    {
      while (bp != NULL && found == NOT_FIND)
        {
          if (strcmp(key, bp->symbol) == MATCH)
              found = FOUND;
          else
              bp = bp->link;
        }
    }

  /* try to renew old symbol, give error message */

  if (found == FOUND && poption == NEW_SYM)
    {
      return NULL;
    }

  /* try to access non-existing symbol, give error message */

  if (found == NOT_FIND && poption == OLD_SYM)
    {
      return NULL;
    }

  /* got the symbol, then return it if in the same class */
```

```
  /* otherwise give error message */

  if (found == FOUND && poption == IND_SYM)
    {
      return (bp);
    }

  /* create the symbol, then return it */

  if (found == NOT_FIND && poption <= IND_SYM)
    {
      switch(pclass)
        {
          case T_SYM: nsyms++; break;
          case T_FUNC: nfuncs++; break;
          case T_SET_BF: nsbfs++; break;
          case T_SET_PC: nspcs++; break;
          case T_SET_MV: nsmvs++; break;
          case T_SET_RP: nsrps++; break;
          case T_SET_RV: nsrvs++; break;
          case T_MODEL: nsmods++; break;
          case T_EL_SET_BF: nebfs++; break;
          case T_EL_SET_PC: nepcs++; break;
          case T_EL_SET_MV: nemvs++; break;
          case T_EL_SET_RP: nerps++; break;
          case T_EL_SET_RV: nervs++; break;
          case T_EL_MODEL: nemods++; break;
          case T_CON_MODEL: ncmods++; break;
        }

      bp = NEW(bucket);
      bp->link = symtab[hashval];
      bp->next = NULL;
      bp->symbol = copys(key);
      bp->form_symbol = transform(key);
      bp->class = pclass;
      bp->scope = scope;
      bp->expr = NULL;

      if (firstsymbol == NULL)
        {
          firstsymbol = bp;
          lastsymbol = bp;
        }
      else
        {
          lastsymbol->next = bp;
          lastsymbol = bp;
        }

      symtab[hashval] = bp;
      return (bp);
    }

}

node *putnode(node *left,short token,node *right)
{
  node *n;
```

```
  n = NEW(node);
  n->value = 0;
  n->type = token;
  n->argument = NULL;
  n->left = left;
  n->right = right;
  return (n);
}

arg *putarg(node *n)
{
  arg *argument;

  argument = NEW(arg);
  argument->argexpr = n;
  argument->argnext = NULL;
  return (argument);
}

skiplist *putskiplist(bucket *b)
{
  skiplist *sl;

  sl = NEW(skiplist);
  sl->membr = b;
  sl->skiplistnext = NULL;
  return (sl);
}

char *printtree(node *n)
{
  register arg *a;
  register bucket *b;
  register short bracket;
  register char *s;

  s = (char *) malloc(MAX_LENGTH);
  bracket = NO_BRACKET;
  switch(n->type)
    {
      case T_NUM:  strcat(s,n->form_value);
                   break;
      case T_SYM:  b = n->buck;
                   strcat(s,b->form_symbol);
                   break;
      case T_FUNC: b = n->buck;
                   strcat(s,b->form_symbol);
                   strcat(s,"(");
                   for (a = n->argument; a->argnext; a = a->argnext)
                      { strcat(s,printtree(a->argexpr));
                        strcat(s,",");
                      }
                   strcat(s,printtree(a->argexpr));
                   strcat(s,")");
                   break;
      case '+':    strcat(s,printtree(n->left));
                   strcat(s," + ");
                   strcat(s,printtree(n->right));
```

```
                    break;
        case '-':   if (n->left->type != T_NUM || n->left->value != NULL)
                      { strcat(s,printtree(n->left));
                      }
                    strcat(s," - ");
                    strcat(s,printtree(n->right));
                    break;
        case '*':   if (n->left->type > T_NUM)
                        bracket = PUT_BRACKET;
                    if (bracket) { strcat(s,"("); }
                    strcat(s,printtree(n->left));
                    if (bracket) { strcat(s,")"); }
                    strcat(s,"*"); bracket = NO_BRACKET;
                    if (n->right->type > T_NUM)
                        bracket = PUT_BRACKET;
                    if (bracket) { strcat(s,"("); }
                    strcat(s,printtree(n->right));
                    if (bracket) { strcat(s,")"); }
                    break;
        case '/':   if (n->left->type > T_NUM)
                        bracket = PUT_BRACKET;
                    if (bracket) { strcat(s,"("); }
                    strcat(s,printtree(n->left));
                    if (bracket) { strcat(s,")"); }
                    strcat(s,"/"); bracket = NO_BRACKET;
                    if (n->right->type > T_NUM)
                        bracket = PUT_BRACKET;
                    if (bracket) { strcat(s,"("); }
                    strcat(s,printtree(n->right));
                    if (bracket) { strcat(s,")"); }
                    break;
        case '^':   if (n->left->type > T_NUM)
                        bracket = PUT_BRACKET;
                    if (bracket) { strcat(s,"("); }
                    strcat(s,printtree(n->left));
                    if (bracket) { strcat(s,")"); }
                    strcat(s,"^"); bracket = NO_BRACKET;
                    if (n->right->type > T_NUM)
                        bracket = PUT_BRACKET;
                    if (bracket) { strcat(s,"("); }
                    strcat(s,printtree(n->right));
                    if (bracket) { strcat(s,")"); }
                    break;
    }
  return (s);
}

char *add_index(char *symbol,char *number)
{
  register char *s;

  s = (char *) malloc(strlen(symbol)+strlen(number)+8);
  strcpy(s,symbol);
  strcat(s,"[");
  strcat(s,number);
  strcat(s,"]");
  return (s);

}
```

```
char *add_form_index(char *symbol,char *number)
{
  register char *s;
  register char *result;

  s = add_index(symbol,number);
  result = transform(s);
  return (result);
}


bucket *add_element(bucket *set,short set_element,short scope,
                    short class,short option)
{
  register char *form_symbol;
  register char *new_symbol;
  register char *new_element;
  register char *number;
  register char **r;
  register bucket *b;

  form_symbol = set->form_symbol;
  number = (char *) malloc(MAX_LENGTH);
  sprintf(number,"%d",set_element);
  new_symbol = add_index(set->symbol,number);
  if (strcmp(form_symbol,set->symbol) == MATCH)
    { new_element = (char *) malloc(strlen(form_symbol)+
                                    strlen(number)+8);
      strcpy(new_element,"[");
      strcat(new_element,form_symbol);
      strcat(new_element,"$<");
      strcat(new_element,number);
      strcat(new_element,"$>]");
    }
  else
    { r = get_name_and_order(set->symbol);
      new_element = (char *) malloc(strlen(r[0])+
                                    strlen(r[1])+
                                    strlen(number)+16);
      strcpy(new_element,"[");
      strcat(new_element,r[0]);
      strcat(new_element,"$<");
      strcat(new_element,r[1]);
      strcat(new_element,"$>$<");
      strcat(new_element,number);
      strcat(new_element,"$>]");
    }
  b = putsym(new_symbol,scope,class,option);
  if (b != NULL)
    { b->form_symbol = new_element; }
  return (b);

}

void free_scope(short scope)
{
  register bucket *bp,*bptmp;
  register short fscope;
```

```
    register int i;

    fscope = scope;

    for (i = 0; i < TABSIZE; i++)
      {
         bp = symtab[i];
         while (bp && bp->scope == fscope)
           {
             bptmp = bp->link;
             free(bp);
             bp->scope = NULL;
             bp->symbol = (char *)0;
             bp->form_symbol = (char *)0;
             switch(bp->class)
               {
                  case T_SYM: nsyms--; break;
                  case T_FUNC: nfuncs--; break;
                  case T_SET_BF: nsbfs--; break;
                  case T_SET_PC: nspcs--; break;
                  case T_SET_MV: nsmvs--; break;
                  case T_SET_RP: nsrps--; break;
                  case T_SET_RV: nsrvs--; break;
                  case T_MODEL: nsmods--; break;
                  case T_EL_SET_BF: nebfs--; break;
                  case T_EL_SET_PC: nepcs--; break;
                  case T_EL_SET_MV: nemvs--; break;
                  case T_EL_SET_RP: nerps--; break;
                  case T_EL_SET_RV: nervs--; break;
                  case T_EL_MODEL: nemods--; break;
                  case T_CON_MODEL: ncmods--; break;
               }
             bp->class = NULL;
             bp->expr = NULL;
             bp = NO_BUCK;
             bp = bptmp;
           }
         if (bp)
            symtab[i] = bp;
         else
            { free(symtab[i]); symtab[i] = NO_BUCK; }
      }
    }
```

E.5 Routine for generating input file to FORM

The following code is used for generating input file to **FORM** based on the description of stochatic model written in the prototype of DEMM language environment.

File AppendixE/generateform.c

```
void generate_form(FILE *form_file)
{
```

```
register bucket *b,*c,*d,*element_bucket;
register skiplist *skip_list,*first_skip,*list;
register short i,j,k,l,count,number_element,n_element;
register char *n = "n_f_ex_value_1x1";
register char *number,*element,*element_bucket_expr;
register char *element_bucket_form,*related_set;

/* print header of form file */

fprintf(form_file,"#-");
fprintf(form_file,"\nNwrite \t statistics;");
fprintf(form_file,"\nformat \t fortran;\n");

/* declare all symbols used in form file */

fprintf(form_file,"\n* declare variables needed for polynomial chaos expansion\n");

/* symbols */

count = 1;
fprintf(form_file,"\nSymbols \t%s",n);
for (i = 0; i < TABSIZE; i++)
   { b = symtab[i];
     while (b != NULL)
       { if (b->class == T_SYM)
           { if (b->scope == scope) /* first in this scope */
               { count++;
                 switch(count)
                   { case FIRST: fprintf(form_file,
                     "\nSymbols \t%s",b->form_symbol);
                     break;
                     case NSYMBOLLINE: fprintf(form_file,
                     ",%s;",b->form_symbol);
                     count = 0;
                     break;
                     default: fprintf(form_file,
                     ",%s",b->form_symbol);
                     break;
                   }
               }
             else
               { c = putsym(b->symbol,scope,T_SYM,OLD_SYM);
                 if (c->scope != scope) /* then global scope */
                   { count++;
                     switch(count)
                       { case FIRST: fprintf(form_file,
                         "\nSymbols \t%s",b->form_symbol);
                         break;
                         case NSYMBOLLINE: fprintf(form_file,
                         ",%s;",b->form_symbol);
                         count = 0;
                         break;
                         default: fprintf(form_file,
                         ",%s",b->form_symbol);
                         break;
                       }
                   }
               }
           }
```

```
            b = b->link;

        }
    }

/* put the last semi colon whenever appropriate */
if (count < NSYMBOLLINE && count != 0)
     fprintf(form_file,";");

/* functions */

count = 0;
for (i = 0; i < TABSIZE; i++)
   { b = symtab[i];
     while (b != NULL)
       { if (b->class == T_FUNC)
           { if (b->scope == scope) /* first in this scope */
               { count++;
                 switch(count)
                   { case FIRST: fprintf(form_file,
                     "\nCFunctions \t%s",b->form_symbol);
                     break;
                     case NSYMBOLLINE: fprintf(form_file,
                     ",%s;",b->form_symbol);
                     count = 0;
                     break;
                     default: fprintf(form_file,
                     ",%s",b->form_symbol);
                     break;
                   }
               }
             else
               { c = putsym(b->symbol,scope,T_FUNC,OLD_SYM);
                 if (c->scope != scope) /* then global scope */
                   { count++;
                     switch(count)
                       { case FIRST: fprintf(form_file,
                         "\nCFunctions \t%s",b->form_symbol);
                         break;
                         case NSYMBOLLINE: fprintf(form_file,
                         ",%s;",b->form_symbol);
                         count = 0;
                         break;
                         default: fprintf(form_file,
                         ",%s",b->form_symbol);
                         break;
                       }
                   }
               }
           }

           b = b->link;

       }
    }

 /* put the last semi colon whenever appropriate */
if (count < NSYMBOLLINE && count != 0)
```

```
    fprintf(form_file,";");

/* sets BF */

for (i = 0; i < TABSIZE; i++)
   { b = symtab[i];
     while (b != NULL)
       { if (b->class == T_SET_BF)
           { if (b->scope == scope) /* first in this scope */
               { count = 0;
                 fprintf(form_file,
                 "\nSet \t\t%s:",b->form_symbol);
                 number_element = b->number_element;
                 for (j = 1; j <= number_element; j++)
                    { number = (char *) malloc(MAX_LENGTH);
                      sprintf(number,"%d",j);
                      element = add_index(b->symbol,number);
                      /* get every element */
                      element_bucket = putsym(element,scope,
                      T_EL_SET_BF,OLD_SYM);
                      element_bucket_expr = printtree(
                      element_bucket->expr);
                      count++;
                      switch(count)
                        { case FIRST: fprintf(form_file,
                          "%s",element_bucket_expr);
                          break;
                          case NSYMBOLLINE:
                          if (j != number_element)
                            { fprintf(
                          form_file,",%s,\n\t\t",
                          element_bucket_expr); }
                          else
                            { fprintf(
                          form_file,",%s;",
                          element_bucket_expr); }
                          count = 0;
                          break;
                          default: fprintf(form_file,
                          ",%s",element_bucket_expr);
                          break;
                        }
                    }

                  /* put the last semi colon
                     whenever appropriate */
                  if (count < NSYMBOLLINE && count != 0)
                  fprintf(form_file,";");

               }
             else
               { c = putsym(b->symbol,scope,T_SET_BF,OLD_SYM);
                 if (c->scope != scope) /* then global scope */
                  { count = 0;
                    fprintf(form_file,
                    "\nSet \t\t%s:",b->form_symbol);
                    number_element = b->number_element;
                    for (j = 1; j <= number_element; j++)
                       { number = (char *) malloc(MAX_LENGTH);
```

```
                        sprintf(number,"%d",j);
                        element = add_index(b->symbol,number);
                        /* get every element */
                        element_bucket = putsym(element,
                        scope-1,T_EL_SET_BF,OLD_SYM);
                        element_bucket_expr = printtree(
                        element_bucket->expr);
                        count++;
                        switch(count)
                          { case FIRST: fprintf(form_file,
                            "%s",element_bucket_expr);
                            break;
                            case NSYMBOLLINE:
                            if (j != number_element)
                              { fprintf(
                            form_file,",%s,\n\t\t",
                            element_bucket_expr); }
                            else
                              { fprintf(
                            form_file,",%s;",
                            element_bucket_expr); }
                            count = 0;
                            break;
                            default: fprintf(form_file,
                            ",%s",element_bucket_expr);
                            break;
                          }
                      }

                    /* put the last semi colon
                       whenever appropriate */
                    if (count < NSYMBOLLINE && count != 0)
                    fprintf(form_file,";");
                  }
              }
          }

        b = b->link;

      }
  }

/* sets MV */
/* first put the symbols */

for (i = 0; i < TABSIZE; i++)
  { b = symtab[i];
    while (b != NULL)
      { if (b->class == T_SET_MV)
          { if (b->scope == scope) /* first in this scope */
              { count = 0;
                fprintf(form_file,"\nSymbols \t");
                number_element = b->number_element;
                for (j = 1; j <= number_element; j++)
                   { number = (char *) malloc(MAX_LENGTH);
                     sprintf(number,"%d",j);
                     element = add_index(b->symbol,number);
                     /* get every element */
                     element_bucket = putsym(element,scope,
```

```
            T_EL_SET_MV,OLD_SYM);
            element_bucket_form =
            element_bucket->form_symbol;
            count++;
            switch(count)
              { case FIRST: fprintf(form_file,
                "%s",element_bucket_form);
                            break;
                case NSYMBOLLINE:
                if (j != number_element)
                                  { fprintf(
                form_file,",%s,\n\t\t",
                element_bucket_form); }
                                else
                                  { fprintf(
                form_file,",%s;",
                element_bucket_form); }
                                count = 0;
                                break;
                default: fprintf(form_file,",%s",
                element_bucket_form);
                         break;
              }
           }

          /* put the last semi colon
             whenever appropriate */
          if (count < NSYMBOLLINE && count != 0)
          fprintf(form_file,";");

        }
      else
        { c = putsym(b->symbol,scope,T_SET_MV,OLD_SYM);
          if (c->scope != scope) /* then global scope */
           { count = 0;
             fprintf(form_file,"\nSymbols \t");
             number_element = b->number_element;
             for (j = 1; j <= number_element; j++)
                { number = (char *) malloc(MAX_LENGTH);
                  sprintf(number,"%d",j);
                  element = add_index(b->symbol,number);
                  /* get every element */
                  element_bucket = putsym(element,
                  scope-1,T_EL_SET_MV,OLD_SYM);
                  element_bucket_form =
                  element_bucket->form_symbol;
                  count++;
                  switch(count)
                    { case FIRST: fprintf(form_file,
                      "%s",element_bucket_form);
                                  break;
                      case NSYMBOLLINE:
                      if (j != number_element)
                                        { fprintf(
                      form_file,",%s,\n\t\t",
                      element_bucket_form); }
                                      else
                                        { fprintf(
                      form_file,",%s;",
```

```
                        element_bucket_form); }
                                        count = 0;
                                        break;
                        default: fprintf(form_file,
                        ",%s",element_bucket_form);
                                break;
                    }
                  }

                /* put the last semi colon
                   whenever appropriate */
                if (count < NSYMBOLLINE && count != 0)
                fprintf(form_file,";");
              }
            }
          }

        b = b->link;

      }
  }

/* then the sets */

for (i = 0; i < TABSIZE; i++)
  { b = symtab[i];
    while (b != NULL)
      { if (b->class == T_SET_MV)
          { if (b->scope == scope) /* first in this scope */
              { count = 0;
                fprintf(form_file,
                "\nSet \t\t%s:",b->form_symbol);
                number_element = b->number_element;
                for (j = 1; j <= number_element; j++)
                   { number = (char *) malloc(MAX_LENGTH);
                     sprintf(number,"%d",j);
                     element = add_index(b->symbol,number);
                     /* get every element */
                     element_bucket = putsym(element,scope,
                     T_EL_SET_MV,OLD_SYM);
                     element_bucket_form =
                     element_bucket->form_symbol;
                     count++;
                     switch(count)
                       { case FIRST: fprintf(form_file,
                         "%s",element_bucket_form);
                                   break;
                         case NSYMBOLLINE:
                         if (j != number_element)
                                        { fprintf(
                         form_file,",%s,\n\t\t",
                         element_bucket_form); }
                                     else
                                       { fprintf(
                         form_file,",%s;",
                         element_bucket_form); }
                                     count = 0;
                                     break;
                         default: fprintf(form_file,
```

```
                ",%s",element_bucket_form);
                        break;
                    }
                 }

              /* put the last semi colon
                 whenever appropriate */
              if (count < NSYMBOLLINE && count != 0)
              fprintf(form_file,";");

            }
          else
            { c = putsym(b->symbol,scope,T_SET_MV,OLD_SYM);
              if (c->scope != scope) /* then global scope */
                { count = 0;
                  fprintf(form_file,
                  "\nSet \t\t%s:",b->form_symbol);
                  number_element = b->number_element;
                  for (j = 1; j <= number_element; j++)
                     { number = (char *) malloc(MAX_LENGTH);
                       sprintf(number,"%d",j);
                       element = add_index(b->symbol,number);
                       /* get every element */
                       element_bucket = putsym(element,
                       scope-1,T_EL_SET_MV,OLD_SYM);
                       element_bucket_form =
                       element_bucket->form_symbol;
                       count++;
                       switch(count)
                         { case FIRST: fprintf(form_file,
                           "%s",element_bucket_form);
                                       break;
                           case NSYMBOLLINE:
                           if (j != number_element)
                                            { fprintf(
                           form_file,",%s,\n\t\t",
                           element_bucket_form); }
                                          else
                                            { fprintf(
                           form_file,",%s;",
                           element_bucket_form); }
                                          count = 0;
                                          break;
                           default: fprintf(form_file,
                           ",%s",element_bucket_form);
                                    break;
                         }
                      }

                  /* put the last semi colon
                     whenever appropriate */
                  if (count < NSYMBOLLINE && count != 0)
                  fprintf(form_file,";");
                }
            }
        }

    b = b->link;
```

```
        }
    }


/* define all random parameters used in form file */

fprintf(form_file,"\n\n* define stochastic parameters and
        variables in the problem\n");

/* parameters */

skip_list = NEW(skiplist);
first_skip = NULL;

for (i = 0; i < TABSIZE; i++)
   { b = symtab[i];
     while (b != NULL)
       { if (b->class == T_SET_RP)
           { if (b->scope == scope) /* first in this scope */
               { fprintf(form_file,"\nGlobal \t%s =
                 \n%s;",b->form_symbol,printtree(b->expr));
                 if (first_skip == NULL)
                   { skip_list = putskiplist(b);
                     first_skip = skip_list;
                   }
                 else
                   { skip_list->skiplistnext = putskiplist(b);
                     skip_list = skip_list->skiplistnext;
                   }
               }
             else
               { c = putsym(b->symbol,scope,T_SET_RP,OLD_SYM);
                 if (c->scope != scope) /* then global scope */
                   { fprintf(form_file,"\nGlobal \t%s =
                     \n%s;",b->form_symbol,printtree(b->expr));
                     if (first_skip == NULL)
                       { skip_list = putskiplist(b);
                         first_skip = skip_list;
                       }
                     else
                       { skip_list->skiplistnext =
                         putskiplist(b);
                         skip_list = skip_list->skiplistnext;
                       }
                   }
               }
           }

         b = b->link;

       }
   }

/* variables */

for (i = 0; i < TABSIZE; i++)
   { b = symtab[i];
     while (b != NULL)
```

```
      { if (b->class == T_SET_RV)
          { if (b->scope == scope) /* first in this scope */
              { fprintf(form_file,"\nGlobal \t%s =
                \n%s;",b->form_symbol,printtree(b->expr));
                if (first_skip == NULL)
                  { skip_list = putskiplist(b);
                    first_skip = skip_list;
                  }
                else
                  { skip_list->skiplistnext = putskiplist(b);
                    skip_list = skip_list->skiplistnext;
                  }
              }
            else
              { c = putsym(b->symbol,scope,T_SET_RV,OLD_SYM);
                if (c->scope != scope) /* then global scope */
                  { fprintf(form_file,"\nGlobal \t%s =
                    \n%s;",b->form_symbol,printtree(b->expr));
                    if (first_skip == NULL)
                      { skip_list = putskiplist(b);
                        first_skip = skip_list;
                      }
                    else
                      { skip_list->skiplistnext =
                        putskiplist(b);
                        skip_list = skip_list->skiplistnext;
                      }
                  }
              }
          }

        b = b->link;

      }
  }

/* define all stochastic models used in form file */

fprintf(form_file,"\n\n* define stochastic models\n");

for (i = 0; i < TABSIZE; i++)
  { b = symtab[i];
    while (b != NULL)
      { if (b->class == T_MODEL)
          { if (b->scope == scope) /* first in this scope */
              { number_element = b->number_element;
                for (j = 1; j <= number_element; j++)
                  { fprintf(form_file,"\n.sort\n");
                    number = (char *) malloc(MAX_LENGTH);
                    sprintf(number,"%d",j);
                    element = add_index(b->symbol,number);
                    /* get every element */
                    element_bucket = putsym(element,
                    scope,T_EL_MODEL,OLD_SYM);
                    element_bucket_form = (char *)
                    malloc(MAX_LENGTH);
                    strcpy(element_bucket_form,
                    element_bucket->expr->buck->form_symbol);
                    element_bucket_expr = printtree(
```

```
element_bucket->expr->buck->expr);
fprintf(form_file,"\nLocal \t%s = \n%s;",
element_bucket_form,element_bucket_expr);
list = first_skip;
count = 0;
while(list != NULL)
  { count++;
    switch(count)
      { case FIRST: fprintf(form_file,
        "\nSkip \t%s",
        list->membr->form_symbol);
                    break;
        case NSYMBOLLINE: fprintf(
        form_file,",%s;",
        list->membr->form_symbol);
                          count = 0;
                          break;
        default: fprintf(form_file,
        ",%s",list->membr->form_symbol);
                 break;
      }
    list = list->skiplistnext;
  }
/* put the last semi colon
   whenever appropriate */
if (count < NSYMBOLLINE && count != 0)
    fprintf(form_file,";");
if (first_skip == NULL)
  { skip_list = putskiplist(
    element_bucket->expr->buck);
    first_skip = skip_list;
  }
else
  { skip_list->skiplistnext = putskiplist(
    element_bucket->expr->buck);
    skip_list = skip_list->skiplistnext;
  }

/* print the deterministic equivalent
   model */
fprintf(form_file,"\n\n** print the
deterministic equivalent model \n");
fprintf(form_file,"\nprint \t%s;",
element_bucket_form);

/* do variational approach */
fprintf(form_file,"\n\n** do variational
approach\n");

for (k = 0; k < TABSIZE; k++)
   { d = symtab[k];
     while (d != NULL)
       { if (d->class == T_SET_BF)
           { if (d->scope == scope)
             /* first in this scope */
               { n_element =
                 d->number_element;
                 related_set =
                 d->related_symbol;
```

```
                    for (l = 1; l <=
                    n_element; l++)
                       { number = (char *)
                         malloc(MAX_LENGTH);
                         sprintf(number,
                         "%d",l);
                         element = add_index(
                         d->symbol,number);
                       /* get every element */
                         element_bucket = putsym(element,scope,
                         T_EL_SET_BF,OLD_SYM);
                         element_bucket_expr = printtree(
                         element_bucket->expr);
                         fprintf(form_file,
                         "\nid \t%s\?%s\~%s\?
                         = %s\[%s\];",
                         element_bucket_expr,d->form_symbol,n,
                         related_set,n);
                       }
                  }
                else
                  { c = putsym(d->symbol,
                    scope,T_SET_BF,OLD_SYM);
                    if (c->scope != scope)
                    /* then global scope */
                      { n_element =
                        d->number_element;
                        related_set =
                        d->related_symbol;
                        for (l = 1; l <=
                        n_element; l++)
                           { number = (char *)
                           malloc(MAX_LENGTH);
                             sprintf(number,
                             "%d",l);
                             element = add_index(d->symbol,
                             number);
                           /* get every element */
                             element_bucket =
                             putsym(element,
                             scope,T_EL_SET_BF,
                             OLD_SYM);
                             element_bucket_expr =
                             printtree(
                             element_bucket->expr);
                             fprintf(form_file,
                             "\nid \t%s\?%s\~%s\?
                             = %s\[%s\];",
                             element_bucket_expr,d->form_symbol,n,
                             related_set,n);
                           }
                      }
                  }
              }

            d = d->link;

        }
    }
```

```
for (k = 0; k < TABSIZE; k++)
   { d = symtab[k];
     while (d != NULL)
       { if (d->class == T_SET_MV)
           { if (d->scope == scope)
             /* first in this scope */
               { n_element = d->number_element;
                 related_set = d->related_symbol;
                 for (l = 1; l <= n_element; l++)
                    { number = (char *)
                      malloc(MAX_LENGTH);
                      sprintf(number,"%d",l);
                      element =
                      add_index(d->symbol,number);
                       /* get every element */
                      element_bucket = putsym(
                      element,scope,T_EL_SET_MV,OLD_SYM);
                      element_bucket_form =
                      element_bucket->form_symbol;
                      element_bucket_expr = printtree(
                      element_bucket->expr);
                      fprintf(form_file,"\nid \t%s = %s;",
                              element_bucket_form,
                              element_bucket_expr);
                    }
               }
             else
               { c = putsym(d->symbol,scope,
                     T_SET_MV,OLD_SYM);
                 if (c->scope != scope)
                     /* then global scope */
                   { n_element = d->number_element;
                     related_set = d->related_symbol;
                     for (l = 1; l <= n_element; l++)
                        { number = (char *) malloc(
                          MAX_LENGTH);
                          sprintf(number,"%d",l);
                          element = add_index(
                          d->symbol,number);
                          /* get every element */
                          element_bucket =
                          putsym(element,scope,
                          T_EL_SET_MV,OLD_SYM);
                          element_bucket_form =
                          element_bucket->form_symbol;
                          element_bucket_expr =
                          printtree(element_bucket->expr);
                          fprintf(form_file,
                          "\nid \t%s = %s;",
                          element_bucket_form,
                          element_bucket_expr);
                        }
                    }
               }
           }

         d = d->link;
```

```
        }
      }

  for (k = 0; k < TABSIZE; k++)
     { d = symtab[k];
       while (d != NULL)
         { if (d->class == T_SET_MV)
             { if (d->scope == scope)
                   /* first in this scope */
               { n_element = d->number_element;
                 related_set = d->related_symbol;
                 for (l = 1; l <= n_element; l++)
                    { number = (char *) malloc(
                      MAX_LENGTH);
                      sprintf(number,"%d",l);
                      element = add_index(
                      d->symbol,number);
                      /* get every element */
                      element_bucket = putsym(
                      element,scope,T_EL_SET_MV,OLD_SYM);
                      element_bucket_form =
                      element_bucket->form_symbol;
                      fprintf(form_file,"\nDrop \t%s;",
                      element_bucket_form);
                    }
                 }
               else
                 { c = putsym(d->symbol,scope,T_SET_MV,
                             OLD_SYM);
                   if (c->scope != scope)
                       /* then global scope */
                     { n_element = d->number_element;
                       related_set = d->related_symbol;
                       for (l = 1; l <= n_element; l++)
                          { number = (char *) malloc(
                            MAX_LENGTH);
                            sprintf(number,"%d",l);
                            element = add_index(
                            d->symbol,number);
                            /* get every element */
                            element_bucket =
                            putsym(element,scope,
                            T_EL_SET_MV,OLD_SYM);
                            element_bucket_form =
                            element_bucket->form_symbol;
                            fprintf(form_file,"\nDrop \t%s;",
                            element_bucket_form);
                          }
                     }
                 }
             }

           d = d->link;

         }
     }

  fprintf(form_file,"\n");
}
```

```
                }

              }

            b = b->link;

          }
      }

  fprintf(form_file,"\n.end\n");
}
```

E.6 Routine for calling FORM

The following code is used for calling **FORM** to process an input file generated by the prototype of DEMM language environment.

File AppendixE/check.c

```
short check(char *form_name,char *form_out,char *format_file)
{
     char *form_command,*trto_command,*rm_command;
     FILE *filein;
     int ch;
     short minus,test;

     form_command = (char *) malloc(MAX_LENGTH);
     strcpy(form_command,FORM);
     strcat(form_command," ");
     strcat(form_command,form_name);
     strcat(form_command," ");
     strcat(form_command,">");
     strcat(form_command," ");
     strcat(form_command,form_out);
     system(form_command);
     filein = fopen(form_out,"r");
     if (!filein)
       { printf("\nCheck error: could not open .form file\n");
         exit(1);
       }
     fseek(filein,0,SEEK_SET);
     minus = 0;
     test = 0;
     do
       { ch = fgetc(filein);
         if (ch == '-' && minus == 0 && test == 0)
           { minus = 1; }
         else
           { if (ch == '-' && minus == 1)
               { minus = 2; }
             else
               { if (ch == '>' && minus == 2)
                   { minus = 3; test = 1; }
                 else
```

```
                    { minus = 0; }
                }
            }
        }
      while (ch != EOF && test == 0);
      fclose(filein);
      if (minus == 3)
        { return (1); }
      else
        { trto_command = (char *) malloc(MAX_LENGTH);
          if (strcmp(format_file,".f") == MATCH)
            { strcpy(trto_command,TRTOF); }
          if (strcmp(format_file,".c") == MATCH)
            { strcpy(trto_command,TRTOC); }
          if (strcmp(format_file,".gms") == MATCH)
            { strcpy(trto_command,TRTOG); }
          if (strcmp(format_file,".ABACUSS") == MATCH)
            { strcpy(trto_command,TRTOA); }
          strcat(trto_command," ");
          strcat(trto_command,form_out);
          system(trto_command);
          rm_command = (char *) malloc(MAX_LENGTH);
          strcpy(rm_command,RM);
          strcat(rm_command," ");
          strcat(rm_command,form_out);
          system(rm_command);
          return (0); }
}
```

E.7 Main routine for demm program

The following code is the main routine for the prototype of DEMM language.

File AppendixE/demm.c

```
#include <stdlib.h>
#include <stdio.h>
#include "new.h"
#include "symtab.h"

int nsyms = 0;
int nfuncs = 0;
int nsbfs = 0,nspcs = 0,nsmvs = 0;
int nsrps = 0,nsrvs = 0,nsmods = 0;
int nebfs = 0,nepcs = 0,nemvs = 0;
int nerps = 0,nervs = 0,nemods = 0;
int ncmods = 0;


FILE *filein,*fileout;
FILE *verbatim_out;
extern FILE *yyin,*yyout;
int lineno = 1;
bucket *b;
```

```
int hashvalue,i;

int main(int argc,char *argv[])
{

     if (argc >= 3)
        {     filein = fopen(argv[1],"r");
              if (!filein) {
                 printf("\ncould not open %s\n",argv[1]);
                 exit(1); }
              yyin = filein;
              fileout = fopen(argv[2],"w");
              if (!fileout) {
                 printf("\ncould not open %s\n",argv[2]);
                 exit(1); }
              verbatim_out = fileout;
        }
      else
        {     printf("\nsupply the files\n");
              exit(1);
        }

      initlex();
      tabinit();
      if (yyparse() == NORMAL) { printf("\nsuccess\n"); }
      return 0;
}
```

E.8 Flex input file for handling include files in Abacuss

The following code is written in **Flex** language environment, and this code is used for generating a scanner for the prototype of DEMM language to process "include" statements for **Abacuss**.

File AppendixE/app.l

```
/*
*  handler for include files in ABACUSS
*  from: "Flex", by G.T. Nicol.
*  modified by: Menner A. Tatang / 4.28.94
*/

%{

#define MAX_NEST        100

extern FILE *yyin,*yyout;

YY_BUFFER_STATE include_stack[MAX_NEST];
int     include_count = -1;

void initlex();
char *strsave(char *);

%}
```

```
alpha           [a-zA-Z]
digit           [0-9]
name            {alpha}("_"|{alpha}|{digit})*
blank           [\t ]*
filename        {name}("."{name})*

%x Normal Include

%%

<Normal>^"@"[Ii](NCLUDE|nclude){blank} { BEGIN Include; }

<Include>'\n' { BEGIN Normal; }

<Include>{filename}  { if (include_count >= MAX_NEST) {
                           printf("\nToo many include files\n");
                           exit(1);
                                                  }
                       if (include_count < 0) {
                           include_stack[++include_count] = YY_CURRENT_BUFFER;
                         } else {
                           include_stack[include_count] = YY_CURRENT_BUFFER;
                         }

                       yyin = fopen(yytext,"r");
                       if (!yyin) {
                           printf("\ncould not open %s\n",yyin);
                           exit(1); }

                       yy_switch_to_buffer(include_stack[++include_count] =
                                           yy_create_buffer(yyin,YY_BUF_SIZE));

                       BEGIN Normal;
                     }

<Include><<EOF>> { printf("\nEOF in include\n");
                   yyterminate();
                 }

<Normal><<EOF>> { if (include_count <= 0) {
                      yyterminate();
                    } else {
                      yy_delete_buffer(include_stack[include_count--]);
                      yy_switch_to_buffer(include_stack[include_count]);
                      BEGIN Normal;
                    }
                }

<Normal>'\n'  { fprintf(yyout,"\n"); }
<Normal>.     { fprintf(yyout,"%c",yytext[0]); }

%%

void initlex()
{
   BEGIN Normal;
}
```

```
char *strsave(char *s)
{
    char *t;

    t = (char *)malloc(strlen(s)+1);
    strcpy(t,s);
    return t;
}

int main(int argc,char *argv[])
{

     if (argc >= 3)
        {     yyin = fopen(argv[1],"r");
              if (!yyin) {
                 printf("\ncould not open %s\n",argv[1]);
                 exit(1); }
              yyout = fopen(argv[2],"w");
              if (!yyout) {
                 printf("\ncould not open %s\n",argv[2]);
                 exit(1); }
         }
      else
         {    printf("\nsupply the files\n");
              exit(1);
         }

      initlex();

      yylex();

}
```

E.9 Flex input file for removing continuation in Abacuss

The following code is written in **Flex** language environment, and this code is used for generating a scanner for the prototype of DEMM language to remove the continuation symbol in the expressions for **Abacuss**.

File AppendixE/cont.l

```
/*
*  remove continuation in ABACUSS
*  developed by: Menner A. Tatang / 4.28.94
*/

%{
#include <string.h>

char *yyname1,*yyname2,*yyname3,*yyname4;
char *yyname12,*yyname34;

void initlex();
char *strsave(char *);
%}
```

```
alpha   [a-zA-Z]
digit   [0-9]
blank   [\t ]*
exp     [Ee][-+]?{digit}+
name    {alpha}("_"|{alpha}|{digit})*
inname  [^")"]+
number  {digit}+|{digit}+"."{digit}*({exp})?|{digit}*"."{digit}+({exp})?|{digit}+{exp}
minus   "-"

%s Normal Semicolon Subsc SSubsc Equal Subeq SSubeq

%%

<Normal>";"         { BEGIN Semicolon; }

<Semicolon>{name}   { yyname1 = strsave(yytext);
                      BEGIN Subsc;
                    }

<Subsc>"("          { BEGIN SSubsc; }

<SSubsc>{inname}    { yyname2 = strsave(yytext);
                      BEGIN Subsc; }

<Subsc>")"          { BEGIN Semicolon; }

<Subsc>.            { BEGIN Semicolon; }

<Semicolon>"="      { BEGIN Equal; }

<Semicolon><<EOF>> { BEGIN Normal; }

<Equal>{number}     { yyname12 = (char *)malloc(strlen(yyname1)+strlen(yyname2)+4);
                      strcpy(yyname12,yyname1);
                      strcat(yyname12,"(");
                      strcat(yyname12,yyname2);
                      strcat(yyname12,")");
                      fprintf(yyout,";\n\n%s = %s",yyname12,yytext);
                      free(yyname1);
                      free(yyname2);
                      free(yyname12);
                      BEGIN Normal;
                    }

<Equal>{minus}      { yyname12 = (char *)malloc(strlen(yyname1)+strlen(yyname2)+4);
                      strcpy(yyname12,yyname1);
                      strcat(yyname12,"(");
                      strcat(yyname12,yyname2);
                      strcat(yyname12,")");
                      fprintf(yyout,";\n\n%s = %s",yyname12,yytext);
                      free(yyname1);
                      free(yyname2);
                      free(yyname12);
                      BEGIN Normal;
                    }

<Equal>{name}       { yyname3 = strsave(yytext);
                      BEGIN Subeq;
```

```
                    }

<Subeq>"("          { BEGIN SSubeq; }

<SSubeq>{inname}    { yyname4 = strsave(yytext);
                      BEGIN Subeq; }

<Subeq>")"          { yyname12 = (char *)malloc(strlen(yyname1)+strlen(yyname2)+4);
                      strcpy(yyname12,yyname1);
                      strcat(yyname12,"(");
                      strcat(yyname12,yyname2);
                      strcat(yyname12,")");
                      yyname34 = (char *)malloc(strlen(yyname3)+strlen(yyname4)+4);
                      strcpy(yyname34,yyname3);
                      strcat(yyname34,"(");
                      strcat(yyname34,yyname4);
                      strcat(yyname34,")");
                      if (strcmp(yyname12,yyname34) != 0) {
                          fprintf(yyout,";\n\n%s = %s",yyname12,yyname34);
                        }
                      free(yyname1);
                      free(yyname2);
                      free(yyname12);
                      free(yyname3);
                      free(yyname4);
                      free(yyname34);
                      BEGIN Normal;
                    }

<Subeq>.            { yyname12 = (char *)malloc(strlen(yyname1)+strlen(yyname2)+4);
                      strcpy(yyname12,yyname1);
                      strcat(yyname12,"(");
                      strcat(yyname12,yyname2);
                      strcat(yyname12,")");
                      yyname34 = (char *)malloc(strlen(yyname3)+strlen(yyname4)+4);
                      strcpy(yyname34,yyname3);
                      strcat(yyname34,"(");
                      strcat(yyname34,yyname4);
                      strcat(yyname34,")");
                      if (strcmp(yyname12,yyname34) != 0) {
                          fprintf(yyout,";\n\n%s = %s",yyname12,yyname34);
                        }
                      free(yyname1);
                      free(yyname2);
                      free(yyname12);
                      free(yyname3);
                      free(yyname4);
                      free(yyname34);
                      BEGIN Normal;
                    }

<Normal><<EOF>>     { fprintf(yyout,";");
                      yyterminate();
                    }
<Normal>'\n'        { fprintf(yyout,"\n"); }
<Normal>.           { fprintf(yyout,"%c",yytext[0]); }

%%
```

```
void initlex()
{
   BEGIN Normal;
}

char *strsave(char *s)
{
    char *t;

    t = (char *)malloc(strlen(s)+1);
    strcpy(t,s);
    return t;
}

int main(int argc,char *argv[])
{

     if (argc >= 3)
        {     yyin = fopen(argv[1],"r");
              if (!yyin) {
                 printf("\ncould not open %s\n",argv[1]);
                 exit(1); }
              yyout = fopen(argv[2],"w");
              if (!yyout) {
                 printf("\ncould not open %s\n",argv[2]);
                 exit(1); }
         }
      else
         {    printf("\nsupply the files\n");
              exit(1);
         }

      initlex();

      yylex();

}
```

E.10 Flex input file for postprocessing Abacuss

The following code is written in **Flex** language environment, and this code is used for generating a scanner for the prototype of DEMM language to postprocess the output from **FORM** such that it can be given to **Abacuss**.

File AppendixE/trtoa.l

```
%{
#include "trtoa.tab.h"
#include <math.h>

#undef yywrap
#define yywrap() (1)
char *strsave(char *);
%}
```

```
alpha   [a-zA-Z]
digit   [0-9]
name    {alpha}("_"|{alpha}|{digit})*
exp     [Ee][-+]?{digit}+
spacel  [\t ]*\n
blank   [\t ]*
number  ("-")?({digit}+|{digit}+"."{digit}*({exp})?|
        {digit}*"."{digit}+({exp})?|{digit}+{exp})
inname  "["{blank}{name}{blank}("$<"{blank}({digit}|{name}|",")*{blank}"$>")*{blank}"]"
innumber "["{blank}{number}{blank}"]"

%%
{spacel}    ;
{blank}     ;
{name}      { yylval.c = strsave(yytext);
              return tNAME; }
{inname}    { yylval.c = strsave(yytext);
              return tINNAME; }
{number}    { yylval.c = strsave(yytext);
              return tNUMBER; }
{innumber}  { yylval.c = strsave(yytext);
              return tINNUMBER; }
.           { return yytext[0]; }
%%

char *strsave(char *s)
{
    char *t;

    t = (char *)malloc(strlen(s)+1);
    strcpy(t,s);
    return t;
}
```

E.11 Bison input file for postprocessing Abacuss

The following code is written in **Bison** language environment, and this code is used for generating a parser for the prototype of DEMM language to postprocess the output from **FORM** such that it can be given to **Abacuss**.

File AppendixE/trtoa.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *appendadd(char *,char *,int *);
char *appendsub(char *,char *,int *);
char *appendmult(char *,char *,int *);
char *appenddiv(char *,char *,int *);
char *appendmin(char *,int *);
char *appendpow(char *,char *,int *);
char *appendbrac(char *,int *);
```

```
char *appendfunc(char *,char *,char *,int *);
char *appendarg(char *,char *,int *);

int  nchars = 0;

int  MAXCHARS = 150;

extern FILE *yyout;
%}

%union {
        char *c;
}

%left '-' '+'
%left '*' '/'
%left '^'
%nonassoc UMINUS

%token <c> tNAME
%token <c> tINNAME
%token <c> tNUMBER
%token <c> tINNUMBER

%type <c> expression
%type <c> slexpression

%%
statement_list: statement
        |       '\n'
        |       statement_list statement
        ;

statement:      tNAME '=' expression
                { fprintf(yyout,"%s = %s;\n\n",$1,$3); }
        |       tINNAME '=' expression
                { fprintf(yyout,"%s = %s;\n\n",$1,$3); }
        ;

expression:     expression '+' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendadd($1,$3,&nchars); }
        |       expression '-' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendsub($1,$3,&nchars); }
        |       expression '*' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendmult($1,$3,&nchars); }
        |       expression '/' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appenddiv($1,$3,&nchars); }
        |       expression '^' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+6);
                  $$ = appendpow($1,$3,&nchars); }
        |       '-' expression %prec UMINUS
                { $$ = (char *)malloc(strlen($2)+4);
                  $$ = appendmin($2,&nchars); }
        |       '(' expression ')'
                { $$ = (char *)malloc(strlen($2)+4);
```

```
                $$ = appendbrac($2,&nchars); }
        |       tNAME '(' expression slexpression ')'
                { $$ = (char *)malloc(strlen($1)+strlen($3)+strlen($4)+5);
                  $$ = appendfunc($1,$3,$4,&nchars); }
        |       tNUMBER
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tINNUMBER
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tNAME
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tINNAME
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        ;

slexpression:   /* empty argument */
                { $$ = (char *)malloc(1);
                  $$ = "\0"; }
        |       ',' expression slexpression
                { $$ = (char *)malloc(strlen($2)+strlen($3)+5);
                  $$ = appendarg($2,$3,&nchars); }
        ;

%%

char *appendadd(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," +\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum," + "); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendsub(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
```

```
        dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
        strcpy(dum,s1);
        if (ncdum >= MAXCHARS)
           { strcat(dum," -\n");
             ncdum = strlen(s2);
           }
        else
           { strcat(dum," - "); }
        strcat(dum,s2);
        *nc = ncdum;
        return dum;
  }

  char *appendmult(char *s1,char *s2,int *nc)
  {
       char *dum;
       int ncdum;

       ncdum = *nc;
       ncdum += strlen(s2);
       dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
       strcpy(dum,s1);
       if (ncdum >= MAXCHARS)
          { strcat(dum," *\n");
            ncdum = strlen(s2);
          }
       else
          { strcat(dum,"*"); }
       strcat(dum,s2);
       *nc = ncdum;
       return dum;
  }

  char *appenddiv(char *s1,char *s2,int *nc)
  {
       char *dum;
       int ncdum;

       ncdum = *nc;
       ncdum += strlen(s2);
       dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
       strcpy(dum,s1);
       if (ncdum >= MAXCHARS)
          { strcat(dum," /\n");
            ncdum = strlen(s2);
          }
       else
          { strcat(dum,"/"); }
       strcat(dum,s2);
       *nc = ncdum;
       return dum;
  }

  char *appendmin(char *s1,int *nc)
  {
       char *dum;
       int ncdum;

       ncdum = *nc;
```

```
     ncdum += strlen(s1);
     dum = (char *)malloc(strlen(s1)+4);
     if (ncdum >= MAXCHARS)
        { strcat(dum," -\n");
          ncdum = strlen(s1);
        }
     else
        { strcpy(dum," - "); }
     strcat(dum,s1);
     *nc = ncdum;
     return dum;
}

char *appendpow(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," ^\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum,"^"); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendbrac(char *s1,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += 2;
     dum = (char *)malloc(strlen(s1)+4);
     strcpy(dum,"(");
     strcat(dum,s1);
     strcat(dum,")");
     *nc = ncdum;
     return dum;
}

char *appendfunc(char *s1,char *s2,char *s3,int *nc)
{
     char *dum;
     int  ncdum;

     ncdum = *nc;
     ncdum += strlen(s1)+strlen(s2)+strlen(s3)+5;
     dum = (char *)malloc(strlen(s1)+strlen(s2)+strlen(s3)+5);
     if (ncdum >= MAXCHARS)
        { strcpy(dum,"\n");
          ncdum = strlen(s1)+strlen(s2)+strlen(s3)+5;
```

```
        }
    strcpy(dum,s1);
    strcat(dum,"(");
    strcat(dum,s2);
    strcat(dum,s3);
    strcat(dum,")");
    *nc = ncdum;
    return dum;
}

char *appendarg(char *s1,char *s2,int *nc)
{
    char *dum;
    int  ncdum;

    ncdum = *nc;
    ncdum += strlen(s1)+strlen(s2)+5;
    dum = (char *)malloc(strlen(s1)+strlen(s2)+5);
    if (ncdum >= MAXCHARS)
       { strcpy(dum,"\n");
         ncdum = strlen(s1)+strlen(s2)+5;
       }
    strcpy(dum,",");
    strcat(dum,s1);
    strcat(dum,s2);
    *nc = ncdum;
    return dum;
}
```

E.12 Main file for postprocessing Abacuss

The following code is the main routine for postprocessing the output from **FORM** such that it can be given to **Abacuss**.

File AppendixE/trtoa.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utility.c"

char *input_file;
char *output_file;
FILE *filein, *fileout;
extern FILE *yyin, *yyout;

void main(int argc, char *argv[])
{
/*
*       verify the arguments
*/
```

```
        if (argc >= 3)
           {    input_file = argv[1];
                output_file = argv[2];
           }
        else
           {    printf("\nuse: trtoa input.file output.file\n");
                nerror("Supply the required names of file");
           }

/*
*       check all files
*/

        if ((filein = fopen(input_file,"r")) == NULL)
          {  nerror("Cannot read the input file"); }

        yyin = filein;

        if ((fileout = fopen(output_file,"w")) == NULL)
          {  nerror("Cannot write the output file"); }

        yyout = fileout;


/*
*       parse the input file and see if it is okey
*/

        if (yyparse() == 0)
          { printf("\nSuccess\n");
            close(filein);
            close(fileout);
          }
        else
          { printf("\nNo Success\n");
            close(filein);
          }

}
```

E.13 Script file for postprocessing Abacuss

The following code is the script file required to transform the array notation to the one recognized by **Abacuss**.

File AppendixE/trtoa.sh

```
#!/bin/sh

nhline=`wc -l $1 | awk '{print $1-2}'`
tail -$nhline $1 > $1.tmp.1
sed 's/''\*\*''/''\^''/g' $1.tmp.1 > $1.tmp.2
awk '{ if (substr($0,6,1) == "+")
         { print "     "substr($0,7) }
       else
```

```
        { print $0 }
    }' $1.tmp.2 > $1.tmp.1

./trtoa.out $1.tmp.1 $1.tmp.2

sed 's/''\['']/''''/g' $1.tmp.2 > $1.tmp.1
sed 's/''\]''/''''/g' $1.tmp.1 > $1.tmp.2
sed 's/''\$\>\$\<''/''\,''/g' $1.tmp.2 > $1.tmp.1
sed 's/''\$\<''/''\(''/g' $1.tmp.1 > $1.tmp.2
sed 's/''\$\>''/''\)''/g' $1.tmp.2 > $1.tmp.1

./cont.out $1.tmp.1 $1.tmp.2

cp $1.tmp.2 $1.ABACUSS

rm -f $1.tmp.1
rm -f $1.tmp.2
```

E.14 Flex input file for postprocessing C

The following code is written in **Flex** language environment, and this code is used for generating a scanner for the prototype of DEMM language to postprocess the output from **FORM** such that it can be given to **C** language.

File AppendixE/trtoc.l

```
%{
#include "trtoc.tab.h"
#include <math.h>

#undef yywrap
#define yywrap() (1)
char *strsave(char *);
%}

alpha   [a-zA-Z]
digit   [0-9]
name    {alpha}("_"|{alpha}|{digit})*
exp     [Ee][-+]?{digit}+
spacel  [\t ]*\n
blank   [\t ]*
number  ("-")?({digit}+|{digit}+"."{digit}*({exp})?|
        {digit}*"."{digit}+({exp})?|{digit}+{exp})
inname  "["{blank}{name}{blank}("$<"{blank}({digit}|{name}|",")*{blank}"$>")*{blank}"]"
innumber "["{blank}{number}{blank}"]"

%%
{spacel}    ;
{blank}     ;
{name}      { yylval.c = strsave(yytext);
              return tNAME; }
{inname}    { yylval.c = strsave(yytext);
              return tINNAME; }
```

```
{number}    { yylval.c = strsave(yytext);
              return tNUMBER; }
{innumber}  { yylval.c = strsave(yytext);
              return tINNUMBER; }
.           { return yytext[0]; }
%%

char *strsave(char *s)
{
    char *t;

    t = (char *)malloc(strlen(s)+1);
    strcpy(t,s);
    return t;
}
```

E.15 Bison input file for postprocessing C

The following code is written in **Bison** language environment, and this code is used for generating a parser for the prototype of DEMM language to postprocess the output from **FORM** such that it can be given to **C** language.

File AppendixE/trtoc.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *appendadd(char *,char *,int *);
char *appendsub(char *,char *,int *);
char *appendmult(char *,char *,int *);
char *appenddiv(char *,char *,int *);
char *appendmin(char *,int *);
char *appendpow(char *,char *,int *);
char *appendbrac(char *,int *);
char *appendfunc(char *,char *,char *,int *);
char *appendarg(char *,char *,int *);

int  nchars = 0;

int  MAXCHARS = 150;

extern FILE *yyout;
%}

%union {
        char *c;
}

%left '-' '+'
%left '*' '/'
%left '^'
%nonassoc UMINUS
```

```
%token <c> tNAME
%token <c> tINNAME
%token <c> tNUMBER
%token <c> tINNUMBER

%type <c> expression
%type <c> slexpression

%%
statement_list: statement
        |       '\n'
        |       statement_list statement
        ;

statement:      tNAME '=' expression
                { fprintf(yyout,"%s = %s;\n\n",$1,$3); }
        |       tINNAME '=' expression
                { fprintf(yyout,"%s = %s;\n\n",$1,$3); }
        ;

expression:     expression '+' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendadd($1,$3,&nchars); }
        |       expression '-' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendsub($1,$3,&nchars); }
        |       expression '*' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendmult($1,$3,&nchars); }
        |       expression '/' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appenddiv($1,$3,&nchars); }
        |       expression '^' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+8);
                  $$ = appendpow($1,$3,&nchars); }
        |       '-' expression %prec UMINUS
                { $$ = (char *)malloc(strlen($2)+4);
                  $$ = appendmin($2,&nchars); }
        |       '(' expression ')'
                { $$ = (char *)malloc(strlen($2)+4);
                  $$ = appendbrac($2,&nchars); }
        |       tNAME '(' expression slexpression ')'
                { $$ = (char *)malloc(strlen($1)+strlen($3)+strlen($4)+5);
                  $$ = appendfunc($1,$3,$4,&nchars); }
        |       tNUMBER
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tINNUMBER
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tNAME
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tINNAME
                { $$ = (char *)malloc(strlen($1)+1);
```

```
                    $$ = $1;
                    nchars += strlen($$); }
        ;

slexpression:   /* empty argument */
                { $$ = (char *)malloc(1);
                  $$ = "\0"; }
        |       ',' expression slexpression
                { $$ = (char *)malloc(strlen($2)+strlen($3)+5);
                  $$ = appendarg($2,$3,&nchars); }
        ;

%%

char *appendadd(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," +\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum," + "); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendsub(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," -\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum," - "); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendmult(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;
```

```
      ncdum = *nc;
      ncdum += strlen(s2);
      dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
      strcpy(dum,s1);
      if (ncdum >= MAXCHARS)
         { strcat(dum," *\n");
           ncdum = strlen(s2);
         }
      else
         { strcat(dum,"*"); }
      strcat(dum,s2);
      *nc = ncdum;
      return dum;
}

char *appenddiv(char *s1,char *s2,int *nc)
{
      char *dum;
      int ncdum;

      ncdum = *nc;
      ncdum += strlen(s2);
      dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
      strcpy(dum,s1);
      if (ncdum >= MAXCHARS)
         { strcat(dum," /\n");
           ncdum = strlen(s2);
         }
      else
         { strcat(dum,"/"); }
      strcat(dum,s2);
      *nc = ncdum;
      return dum;
}

char *appendmin(char *s1,int *nc)
{
      char *dum;
      int ncdum;

      ncdum = *nc;
      ncdum += strlen(s1);
      dum = (char *)malloc(strlen(s1)+4);
      if (ncdum >= MAXCHARS)
         { strcat(dum," -\n");
           ncdum = strlen(s1);
         }
      else
         { strcpy(dum," - "); }
      strcat(dum,s1);
      *nc = ncdum;
      return dum;
}

char *appendpow(char *s1,char *s2,int *nc)
{
      char *dum;
      int ncdum;
```

```
        ncdum = *nc;
        ncdum += strlen(s2) + 6;
        dum = (char *)malloc(strlen(s1)+strlen(s2)+12);
        if (ncdum >= MAXCHARS)
           { strcat(dum,"\npow(");
             ncdum = strlen(s1)+strlen(s2)+6;
           }
        else
           { strcpy(dum,"pow("); }
        strcat(dum,s1);
        strcat(dum,",");
        strcat(dum,"1.0*");
        strcat(dum,s2);
        strcat(dum,")");
        *nc = ncdum;
        return dum;
}

char *appendbrac(char *s1,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += 2;
     dum = (char *)malloc(strlen(s1)+4);
     strcpy(dum,"(");
     strcat(dum,s1);
     strcat(dum,")");
     *nc = ncdum;
     return dum;
}

char *appendfunc(char *s1,char *s2,char *s3,int *nc)
{
     char *dum;
     int  ncdum;

     ncdum = *nc;
     ncdum += strlen(s1)+strlen(s2)+strlen(s3)+5;
     dum = (char *)malloc(strlen(s1)+strlen(s2)+strlen(s3)+5);
     if (ncdum >= MAXCHARS)
        { strcpy(dum,"\n");
          ncdum = strlen(s1)+strlen(s2)+strlen(s3)+5;
        }
     strcpy(dum,s1);
     strcat(dum,"(");
     strcat(dum,s2);
     strcat(dum,s3);
     strcat(dum,")");
     *nc = ncdum;
     return dum;
}

char *appendarg(char *s1,char *s2,int *nc)
{
     char *dum;
     int  ncdum;
```

```
        ncdum = *nc;
        ncdum += strlen(s1)+strlen(s2)+5;
        dum = (char *)malloc(strlen(s1)+strlen(s2)+5);
        if (ncdum >= MAXCHARS)
           { strcpy(dum,"\n");
             ncdum = strlen(s1)+strlen(s2)+5;
           }
        strcpy(dum,",");
        strcat(dum,s1);
        strcat(dum,s2);
        *nc = ncdum;
        return dum;
}
```

E.16 Main file for postprocessing C

The following code is the main routine for postprocessing the output from **FORM** such that it can be given to **C** language.

File AppendixE/trtoc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utility.c"

char *input_file;
char *output_file;
FILE *filein, *fileout;
extern FILE *yyin, *yyout;

void main(int argc, char *argv[])
{
/*
*       verify the arguments
*/

        if (argc >= 3)
           {    input_file = argv[1];
                output_file = argv[2];
           }
        else
           {    printf("\nuse: trtoc input.file output.file\n");
                nerror("Supply the required names of file");
           }

/*
*       check all files
*/

        if ((filein = fopen(input_file,"r")) == NULL)
          {  nerror("Cannot read the input file"); }
```

```
        yyin = filein;

        if ((fileout = fopen(output_file,"w")) == NULL)
          {  nerror("Cannot write the output file"); }

        yyout = fileout;


/*
*       parse the input file and see if it is okey
*/

        if (yyparse() == 0)
          { printf("\nSuccess\n");
            close(filein);
            close(fileout);
          }
        else
          { printf("\nNo Success\n");
            close(filein);
          }

}
```

E.17 Script file for postprocessing C

The following code is the script file required to transform the array notation to the one recognized by C language.

File AppendixE/trtoc.sh

```
#!/bin/sh

cp $1 $1.c

nhline=`wc -l $1 | awk '{print $1-2}'`
tail -$nhline $1 > $1.tmp.1
sed 's/''\*\*''/''\^''/g' $1.tmp.1 > $1.tmp.2
awk '{ if (substr($0,6,1) == "+")
          { print "     "substr($0,7) }
        else
          { print $0 }
     }' $1.tmp.2 > $1.tmp.1

./trtoc.out $1.tmp.1 $1.tmp.2

sed 's/''\['']/''''/g' $1.tmp.2 > $1.tmp.1
sed 's/''\]''/''''/g' $1.tmp.1 > $1.tmp.2
sed 's/''\$\<''/''\['']/g' $1.tmp.2 > $1.tmp.1
sed 's/''\$\>''/''\]''/g' $1.tmp.1 > $1.tmp.2

cp $1.tmp.2 $1.c

rm -f $1.tmp.1
```

```
rm -f $1.tmp.2
```

E.18 Script file for postprocessing Fortran

The following code is the script file required to transform the array notation to the one recognized by **Fortran** language.

File AppendixE/trtof.sh

```
#!/bin/sh

nhline=`wc -l $1 | awk '{print $1-2}'`
tail -$nhline $1 > $1.tmp.1
sed 's/''\['''/''''/g' $1.tmp.1 > $1.tmp.2
sed 's/''\]''/''''/g' $1.tmp.2 > $1.tmp.1
sed 's/''\$\>\$\<''/''\,''/g' $1.tmp.1 > $1.tmp.2
sed 's/''\$\<''/''\(''/g' $1.tmp.2 > $1.tmp.1
sed 's/''\$\>''/''\)''/g' $1.tmp.1 > $1.tmp.2
awk '{ if (substr($0,6,1) == "+")
          { print $0 }
        else
          { print "   "$0 }
     }' $1.tmp.2 > $1.tmp.1

cp $1.tmp.1 $1.f

rm -f $1.tmp.1
rm -f $1.tmp.2
```

E.19 Flex input file for postprocessing GAMS

The following code is written in **Flex** language environment, and this code is used for generating a scanner for the prototype of DEMM language to postprocess the output from **FORM** such that it can be given to **GAMS**.

File AppendixE/trtog.l

```
%{
#include "trtog.tab.h"
#include <math.h>

#undef yywrap
#define yywrap() (1)
char *strsave(char *);
%}

alpha   [a-zA-Z]
```

```
digit   [0-9]
name    {alpha}("_"|{alpha}|{digit})*
exp     [Ee][-+]?{digit}+
spacel  [\t ]*\n
blank   [\t ]*
number  ("-")?({digit}+|{digit}+"."{digit}*({exp})?|
        {digit}*"."{digit}+({exp})?|{digit}+{exp})
inname  "["{blank}{name}{blank}("$<"{blank}({digit}|{name}|",")*{blank}"$>")*{blank}"]"
innumber "["{blank}{number}{blank}"]"

%%
{spacel}    ;
{blank}     ;
{name}      { yylval.c = strsave(yytext);
              return tNAME; }
{inname}    { yylval.c = strsave(yytext);
              return tINNAME; }
{number}    { yylval.c = strsave(yytext);
              return tNUMBER; }
{innumber}  { yylval.c = strsave(yytext);
              return tINNUMBER; }
.           { return yytext[0]; }
%%

char *strsave(char *s)
{
    char *t;

    t = (char *)malloc(strlen(s)+1);
    strcpy(t,s);
    return t;
}
```

E.20 Bison input file for postprocessing GAMS

The following code is written in **Bison** language environment, and this code is used for generating a parser for the prototype of DEMM language to postprocess the output from **FORM** such that it can be given to **GAMS**.

File AppendixE/trtog.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *appendadd(char *,char *,int *);
char *appendsub(char *,char *,int *);
char *appendmult(char *,char *,int *);
char *appenddiv(char *,char *,int *);
char *appendmin(char *,int *);
char *appendpow(char *,char *,int *);
char *appendbrac(char *,int *);
char *appendfunc(char *,char *,char *,int *);
```

```
char *appendarg(char *,char *,int *);

int  nchars = 0;

int  MAXCHARS = 150;

extern FILE *yyout;
%}

%union {
        char *c;
}

%left '-' '+'
%left '*' '/'
%left '^'
%nonassoc UMINUS

%token <c> tNAME
%token <c> tINNAME
%token <c> tNUMBER
%token <c> tINNUMBER

%type <c> expression
%type <c> slexpression

%%
statement_list: statement
        |       '\n'
        |       statement_list statement
        ;

statement:      tNAME '=' expression
                { fprintf(yyout,"%s.. %s =E= 0.0;\n\n",$1,$3); }
        |       tINNAME '=' expression
                { fprintf(yyout,"%s.. %s =E= 0.0;\n\n",$1,$3); }
        ;

expression:     expression '+' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendadd($1,$3,&nchars); }
        |       expression '-' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendsub($1,$3,&nchars); }
        |       expression '*' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appendmult($1,$3,&nchars); }
        |       expression '/' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+4);
                  $$ = appenddiv($1,$3,&nchars); }
        |       expression '^' expression
                { $$ = (char *)malloc(strlen($1)+strlen($3)+8);
                  $$ = appendpow($1,$3,&nchars); }
        |       '-' expression %prec UMINUS
                { $$ = (char *)malloc(strlen($2)+4);
                  $$ = appendmin($2,&nchars); }
        |       '(' expression ')'
                { $$ = (char *)malloc(strlen($2)+4);
                  $$ = appendbrac($2,&nchars); }
```

```
        |       tNAME '(' expression slexpression ')'
                { $$ = (char *)malloc(strlen($1)+strlen($3)+strlen($4)+5);
                  $$ = appendfunc($1,$3,$4,&nchars); }
        |       tNUMBER
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tINNUMBER
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tNAME
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        |       tINNAME
                { $$ = (char *)malloc(strlen($1)+1);
                  $$ = $1;
                  nchars += strlen($$); }
        ;

slexpression:   /* empty argument */
                { $$ = (char *)malloc(1);
                  $$ = "\0"; }
        |       ',' expression slexpression
                { $$ = (char *)malloc(strlen($2)+strlen($3)+5);
                  $$ = appendarg($2,$3,&nchars); }
        ;

%%

char *appendadd(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," +\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum," + "); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendsub(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
```

```
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," -\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum," - "); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendmult(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," *\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum,"*"); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appenddiv(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+4);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," /\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum,"/"); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendmin(char *s1,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s1);
```

```
     dum = (char *)malloc(strlen(s1)+4);
     if (ncdum >= MAXCHARS)
        { strcat(dum," -\n");
          ncdum = strlen(s1);
        }
     else
        { strcpy(dum," - "); }
     strcat(dum,s1);
     *nc = ncdum;
     return dum;
}

char *appendpow(char *s1,char *s2,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += strlen(s2);
     dum = (char *)malloc(strlen(s1)+strlen(s2)+6);
     strcpy(dum,s1);
     if (ncdum >= MAXCHARS)
        { strcat(dum," **\n");
          ncdum = strlen(s2);
        }
     else
        { strcat(dum,"**"); }
     strcat(dum,s2);
     *nc = ncdum;
     return dum;
}

char *appendbrac(char *s1,int *nc)
{
     char *dum;
     int ncdum;

     ncdum = *nc;
     ncdum += 2;
     dum = (char *)malloc(strlen(s1)+4);
     strcpy(dum,"(");
     strcat(dum,s1);
     strcat(dum,")");
     *nc = ncdum;
     return dum;
}

char *appendfunc(char *s1,char *s2,char *s3,int *nc)
{
     char *dum;
     int  ncdum;

     ncdum = *nc;
     ncdum += strlen(s1)+strlen(s2)+strlen(s3)+5;
     dum = (char *)malloc(strlen(s1)+strlen(s2)+strlen(s3)+5);
     if (ncdum >= MAXCHARS)
        { strcpy(dum,"\n");
          ncdum = strlen(s1)+strlen(s2)+strlen(s3)+5;
        }
```

```
        strcpy(dum,s1);
        strcat(dum,"(");
        strcat(dum,s2);
        strcat(dum,s3);
        strcat(dum,")");
        *nc = ncdum;
        return dum;
}

char *appendarg(char *s1,char *s2,int *nc)
{
        char *dum;
        int  ncdum;

        ncdum = *nc;
        ncdum += strlen(s1)+strlen(s2)+5;
        dum = (char *)malloc(strlen(s1)+strlen(s2)+5);
        if (ncdum >= MAXCHARS)
           { strcpy(dum,"\n");
             ncdum = strlen(s1)+strlen(s2)+5;
           }
        strcpy(dum,",");
        strcat(dum,s1);
        strcat(dum,s2);
        *nc = ncdum;
        return dum;
}
```

E.21 Main file for postprocessing GAMS

The following code is the main routine for postprocessing the output from **FORM** such that it can be given to **GAMS**.

File AppendixE/trtog.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utility.c"

char *input_file;
char *output_file;
FILE *filein, *fileout;
extern FILE *yyin, *yyout;

void main(int argc, char *argv[])
{
/*
*       verify the arguments
*/

        if (argc >= 3)
```

```
            {    input_file = argv[1];
                 output_file = argv[2];
            }
          else
            {    printf("\nuse: trtoc input.file output.file\n");
                 nerror("Supply the required names of file");
            }

    /*
    *       check all files
    */

          if ((filein = fopen(input_file,"r")) == NULL)
            {  nerror("Cannot read the input file"); }

          yyin = filein;

          if ((fileout = fopen(output_file,"w")) == NULL)
            {  nerror("Cannot write the output file"); }

          yyout = fileout;


    /*
    *       parse the input file and see if it is okey
    */

          if (yyparse() == 0)
            { printf("\nSuccess\n");
              close(filein);
              close(fileout);
            }
          else
            { printf("\nNo Success\n");
              close(filein);
            }

    }
```

E.22 Script file for postprocessing GAMS

The following code is the script file required to transform the array notation to the one recognized by GAMS.

File AppendixE/trtog.sh

```
#!/bin/sh

nhline=`wc -l $1 | awk '{print $1-2}'`
tail -$nhline $1 > $1.tmp.1
sed 's/''\*\*''/''\^''/g' $1.tmp.1 > $1.tmp.2
awk '{ if (substr($0,6,1) == "+")
          { print "     "substr($0,7) }
       else
         { print $0 }
```

```
    }' $1.tmp.2 > $1.tmp.1

./trtog.out $1.tmp.1 $1.tmp.2

sed 's/''\['''/''''/g' $1.tmp.2 > $1.tmp.1
sed 's/''\]'''/''''/g' $1.tmp.1 > $1.tmp.2
sed 's/''\$\>\$\<''/''\"\,\"''/g' $1.tmp.2 > $1.tmp.1
sed 's/''\$\<''/''\(\"''/g' $1.tmp.1 > $1.tmp.2
sed 's/''\$\>''/''\"\)''/g' $1.tmp.2 > $1.tmp.1

cp $1.tmp.1 $1.gms

rm -f $1.tmp.1
rm -f $1.tmp.2
```

Appendix F

DEMM Files for Examples in Chapter 9

F.1 Simple algebraic equations model

F.1.1 file: fiber.dem

The following code is an input file written in the prototype language of DEMM. This input file describes the stochastic model of fiber process that is represented by a set of algebraic equations.

```
%{
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "nrutil.h"

double *a,*b;
%}

# use c language for output files

%Format c

%%

# two independent Gaussian random variables,
# e1 and e2, are used as coordinates

Set [BF] basis = {e1,e2};

# define the value of moments for those coordinates
# for example the fourth moment is 3

Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135};

# we use six terms for x

Set [PC] pcx = {1,e1,e2,e1*e2,e1^2-1,e2^2-1};

# calculate coefficient matrix
```

```
%{
void coeff(double **c)
{
%}

# and put it into coeffdow.c

Start coeffdow

Set [PC] pca = {1,e2};

Set [RP] a -> pca;

Models   coef1 := a =
          {{c[1][1,...,6]} -> {pcx[1,...,6]}},
         coef2 := a*e1 =
          {{c[2][1,...,6]} -> {pcx[1,...,6]}},
         coef3 := a*e2 =
          {{c[3][1,...,6]} -> {pcx[1,...,6]}},
         coef4 := a*e1*e2 =
          {{c[4][1,...,6]} -> {pcx[1,...,6]}},
         coef5 := a*(e1^2-1) =
          {{c[5][1,...,6]} -> {pcx[1,...,6]}},
         coef6 := a*(e2^2-1) =
          {{c[6][1,...,6]} -> {pcx[1,...,6]}};

End

%{
}
%}

# calculate force vector

%{
void force(double *f)
{
%}

# and put it into forcedow.c

Start forcedow

# we use six terms for b

Set [PC] pcb = {1,e1,e2,e1*e2};

Set [RP] b -> pcb;

Models   force := b =
          {{f[1,...,6]} -> {pcx[1,...,6]}};

End

%{
}
%}
```

```
%%

#define N       6

int main(void)
{
  int i,*indx;
  double d;
  double **c,*f;
  a=dvector(1,2);
  b=dvector(1,6);
  a[1]=1.61946;a[2]=0.000436368;
  b[1]=52.2066;b[2]=0.10072;b[3]=-0.211292;b[4]=-0.000407636;
  indx=ivector(1,N);
  c=dmatrix(1,N,1,N);
  f=dvector(1,N);
  coeff(c);
  force(f);
  ludcmp(c,N,indx,&d);
  lubksb(c,N,indx,f);
  printf("i\tx\n");
  for(i=1;i<=N;i++)
    { printf("%d\t%12.8f\n",i,f[i]); }
  return (0);
}
```

F.2 Climate forcing by anthropogenic aerosols

F.2.1 file: aerosolsl.dem

The following code is an input file written in the prototype language of DEMM. This input file describes the stochastic model of climate forcing that is represented by a set of algebraic equations.

```
# example for algebraic equations
# climate forcing by anthropogenic aerosols model
# see Charlson et.al, Science 24 January 1992 p. 423

%{

/* Example of a simple box model for aerosols */

#include <stdio.h>
#include <math.h>
#define NRANSI
#include "nrutil.h"

/* Define global variables */

double  Q0,Y0,ts0,al0,fh0,T0,ma0,ms0,b0,
        Q1,Y1,ts1,al1,fh1,T1,ma1,ms1,b1,
        Q2,Y2,ts2,al2,fh2,T2,ma2,ms2,b2,
        Q3,Y3,ts3,al3,fh3,T3,ma3,ms3,b3;
double  A,ft,std;

/* Define the model */
```

```
void funcv(int n, double x[], double f[])
{

%}

# define the format of output file

%Format c

# begin the stochastic modeling

%%

# define the coordinates for this problem

Set [BF] basis = {e1,e2,e3,e4,e5,e6,e7,e8,e9};

# declare the values of moments
# in this case, we give the first fourteen

Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135};

# declare the form of polynomial chaos expansion
# since we use Gaussian random variable as the coordinate
# then the polynomial chaos expansion uses Hermite polynomials.

Set [PC] pcc = {1,e1,e2,e3,e4,e5,e6,e7,e8,e9,
                e2*e3,e2*e4,e2*e6,e3*e4,e3*e6,e4*e6,
                e1^2-1,e2^2-1,e3^2-1,
                e4^2-1,e5^2-1,e6^2-1,e7^2-1,e8^2-1,e9^2-1,
                e1^3-3*e1,e2^3-3*e2,e3^3-3*e3,e4^3-3*e4,
                e5^3-3*e5,e6^3-3*e6,e7^3-3*e7,e8^3-3*e8,
                e9^3-3*e9};

# define uncertain parameters, and assume that
# they are independent Gaussian distributed random variables

Set [RP] Q := Q0 + Q1*e1 + Q2*(e1^2-1) + Q3*(e1^3-3*e1),
         Y := Y0 + Y1*e2 + Y2*(e2^2-1) + Y3*(e2^3-3*e2),
         ts := ts0 + ts1*e3 + ts2*(e3^2-1) + ts3*(e3^3-3*e3),
         al := al0 + al1*e4 + al2*(e4^2-1) + al3*(e4^3-3*e4),
         fh := fh0 + fh1*e5 + fh2*(e5^2-1) + fh3*(e5^3-3*e5),
         T := T0 + T1*e6 + T2*(e6^2-1) + T3*(e6^3-3*e6),
         ma := ma0 + ma1*e7 + ma2*(e7^2-1) + ma3*(e7^3-3*e7),
         ms := ms0 + ms1*e8 + ms2*(e8^2-1) + ms3*(e8^3-3*e8),
         b := b0 + b1*e9 + b2*(e9^2-1) + b3*(e9^3-3*e9);

# define dependent variables, and now become
# uncertain variables

Set [RV] Fr -> pcc = {{x[1,...,34]}},
         ds -> pcc = {{x[35,...,68]}},
         bs -> pcc = {{x[69,...,102]}};

# define the name of file for storing the deterministic
# equivalent model

Start aerosolslmodel
```

```
# climate forcing by aerosols model

Models eq1 := Fr + 0.5*ft*T*ma*ms*b*ds =
              {{f[1,...,34]} -> {pcc[1,...,34]}},
       eq2 := ds - al*fh*bs =
              {{f[35,...,68]} -> {pcc[1,...,34]}},
       eq3 := bs - 3.0*Q*Y*ts/A =
              {{f[69,...,102]} -> {pcc[1,...,34]}};

End

# end of the stochastic modeling

%%

}

#define N 102

int main(void)
{

        int i,check;
        double *x,*f;

/*      Define the values of parameters */

        Q0 = 71.6968e12; Q1 = -0.936964e13; Q2 = 0.148212e13; Q3 = 0.12392e13;
        Y0 = 0.542837; Y1 = 0.219130375; Y2 = 0.04814654; Y3 = 0.0;
        ts0 = 1.65867e-2; ts1 = 0.00669567; ts2 = 0.00147115; ts3 = 0.0;
        A = 5e14;
        al0 = 5.2912; al1 = 1.83069843; al2 = 0.14507543; al3 = 0.03420186;
        fh0 = 1.72849; fh1 = 0.31735; fh2 = 0.011663; fh3 = 0.0;
        ft = 1370.0;
        T0 = 0.613779; T1 = -0.176412; T2 = 0.073569; T3 = 0.0231393;
        ma0 = 0.391775; ma1 = 0.0374175; ma2 = 0.000535397; ma3 = 0.0;
        ms0 = 0.732067; ms1 = -0.123296765; ms2 = 0.025718525; ms3 = 0.016274328;
        b0 = 0.310505; b1 = -0.072822779; b2 = 0.022560723; b3 = 0.009567571;

        x=dvector(1,N);
        f=dvector(1,N);

/*      Give initial guesses */

        x[1]=-1.3;x[2]=0.1;x[3]=0.1;x[4]=0.1;x[5]=0.1;
        x[6]=0.1;x[7]=0.1;x[8]=0.1;x[9]=0.1;
        x[10]=0.1;x[11]=0.1;x[12]=0.1;x[13]=0.1;x[14]=0.1;
        x[15]=0.1;x[16]=0.1;x[17]=0.1;x[18]=0.1;
        x[19]=0.1;x[20]=0.1;x[21]=0.1;x[22]=0.1;x[23]=0.1;
        x[24]=0.1;x[25]=0.1;x[26]=0.1;x[27]=0.1;
        x[28]=0.1;x[29]=0.1;x[30]=0.1;x[31]=0.1;
        x[32]=0.1;x[33]=0.1;x[34]=0.1;
        x[35]=0.04;
        x[36]=0.01;x[37]=0.01;x[38]=0.01;
        x[39]=0.01;x[40]=0.01;x[41]=0.01;
        x[42]=0.01;x[43]=0.01;x[44]=0.01;x[45]=0.01;
        x[46]=0.01;x[47]=0.01;x[48]=0.01;x[49]=0.01;
        x[50]=0.01;x[51]=0.01;x[52]=0.01;x[53]=0.01;
```

```
        x[54]=0.01;x[55]=0.01;x[56]=0.01;x[57]=0.01;
        x[58]=0.01;x[59]=0.01;x[60]=0.01;x[61]=0.01;x[62]=0.01;
        x[63]=0.01;x[64]=0.01;x[65]=0.01;x[66]=0.01;
        x[67]=0.01;x[68]=0.01;
        x[69]=0.004;
        x[70]=0.001;x[71]=0.001;
        x[72]=0.001;x[73]=0.001;x[74]=0.001;x[75]=0.001;
        x[76]=0.001;x[77]=0.001;x[78]=0.001;x[79]=0.001;
        x[80]=0.001;x[81]=0.001;x[82]=0.001;x[83]=0.001;
        x[84]=0.001;x[85]=0.001;x[86]=0.001;x[87]=0.001;
        x[88]=0.001;x[89]=0.001;
        x[90]=0.001;x[91]=0.001;x[92]=0.001;x[93]=0.001;
        x[94]=0.001;x[95]=0.001;
        x[96]=0.001;x[97]=0.001;x[98]=0.001;x[99]=0.001;
        x[100]=0.001;x[101]=0.001;x[102]=0.001;


/*      Use BROYDN subroutine from Numerical Recipes book */
/*      by Press et.al */

        broydn(x,N,&check,funcv);

/*      Print the results */

        funcv(N,x,f);
        if (check) printf("Convergence problems.\n");
        printf("%7s %3s %12s\n","Index","x","f");
        for (i=1;i<=N;i++) printf("%5d %12.6f %e\n",i,x[i],f[i]);
        std = 0.0;
        for (i=2;i<=16;i++) std = std + x[i]*x[i];
        for (i=17;i<=25;i++) std = std + 2.0*x[i]*x[i];
        for (i=26;i<=34;i++) std = std + 6.0*x[i]*x[i];
        printf("\nstd.dev. FR = %e\n",sqrt(std));
        std = 0.0;
        for (i=36;i<=50;i++) std = std + x[i]*x[i];
        for (i=51;i<=59;i++) std = std + 2.0*x[i]*x[i];
        for (i=60;i<=68;i++) std = std + 6.0*x[i]*x[i];
        printf("\nstd.dev. delta = %e\n",sqrt(std));
        std = 0.0;
        for (i=70;i<=84;i++) std = std + x[i]*x[i];
        for (i=85;i<=93;i++) std = std + 2.0*x[i]*x[i];
        for (i=94;i<=102;i++) std = std + 6.0*x[i]*x[i];
        printf("\nstd.dev. B = %e\n",sqrt(std));
        free_dvector(f,1,N);
        free_dvector(x,1,N);
        return 0;
}
#undef NRANSI
```

F.2.2 file: aerosolsl.c

The following code is an output file in C language generated by the prototype language of DEMM. This output file is the main routine for the climate forcing calculation.

```
/* Example of a simple box model for aerosols */

#include <stdio.h>
```

```
#include <math.h>
#define NRANSI
#include "nrutil.h"

/* Define global variables */

double  Q0,Y0,ts0,al0,fh0,T0,ma0,ms0,b0,
        Q1,Y1,ts1,al1,fh1,T1,ma1,ms1,b1,
        Q2,Y2,ts2,al2,fh2,T2,ma2,ms2,b2,
        Q3,Y3,ts3,al3,fh3,T3,ma3,ms3,b3;
double  A,ft,std;

/* Define the model */

void funcv(int n, double x[], double f[])
{


#include "aerosolslmodel.c"


}

#define N 102

int main(void)
{

        int i,check;
        double *x,*f;

/*      Define the values of parameters */

        Q0 = 71.6968e12; Q1 = -0.936964e13; Q2 = 0.148212e13; Q3 = 0.12392e13;
        Y0 = 0.542837; Y1 = 0.219130375; Y2 = 0.04814654; Y3 = 0.0;
        ts0 = 1.65867e-2; ts1 = 0.00669567; ts2 = 0.00147115; ts3 = 0.0;
        A = 5e14;
        al0 = 5.2912; al1 = 1.83069843; al2 = 0.14507543; al3 = 0.03420186;
        fh0 = 1.72849; fh1 = 0.31735; fh2 = 0.011663; fh3 = 0.0;
        ft = 1370.0;
        T0 = 0.613779; T1 = -0.176412; T2 = 0.073569; T3 = 0.0231393;
        ma0 = 0.391775; ma1 = 0.0374175; ma2 = 0.000535397; ma3 = 0.0;
        ms0 = 0.732067; ms1 = -0.123296765; ms2 = 0.025718525; ms3 = 0.016274328;
        b0 = 0.310505; b1 = -0.072822779; b2 = 0.022560723; b3 = 0.009567571;

        x=dvector(1,N);
        f=dvector(1,N);

/*      Give initial guesses */

        x[1]=-1.3;x[2]=0.1;x[3]=0.1;x[4]=0.1;x[5]=0.1;
        x[6]=0.1;x[7]=0.1;x[8]=0.1;x[9]=0.1;
        x[10]=0.1;x[11]=0.1;x[12]=0.1;x[13]=0.1;x[14]=0.1;
        x[15]=0.1;x[16]=0.1;x[17]=0.1;x[18]=0.1;
        x[19]=0.1;x[20]=0.1;x[21]=0.1;x[22]=0.1;x[23]=0.1;
        x[24]=0.1;x[25]=0.1;x[26]=0.1;x[27]=0.1;
        x[28]=0.1;x[29]=0.1;x[30]=0.1;x[31]=0.1;
        x[32]=0.1;x[33]=0.1;x[34]=0.1;
        x[35]=0.04;
```

```
        x[36]=0.01;x[37]=0.01;x[38]=0.01;
        x[39]=0.01;x[40]=0.01;x[41]=0.01;
        x[42]=0.01;x[43]=0.01;x[44]=0.01;x[45]=0.01;
        x[46]=0.01;x[47]=0.01;x[48]=0.01;x[49]=0.01;
        x[50]=0.01;x[51]=0.01;x[52]=0.01;x[53]=0.01;
        x[54]=0.01;x[55]=0.01;x[56]=0.01;x[57]=0.01;
        x[58]=0.01;x[59]=0.01;x[60]=0.01;x[61]=0.01;x[62]=0.01;
        x[63]=0.01;x[64]=0.01;x[65]=0.01;x[66]=0.01;
        x[67]=0.01;x[68]=0.01;
        x[69]=0.004;
        x[70]=0.001;x[71]=0.001;
        x[72]=0.001;x[73]=0.001;x[74]=0.001;x[75]=0.001;
        x[76]=0.001;x[77]=0.001;x[78]=0.001;x[79]=0.001;
        x[80]=0.001;x[81]=0.001;x[82]=0.001;x[83]=0.001;
        x[84]=0.001;x[85]=0.001;x[86]=0.001;x[87]=0.001;
        x[88]=0.001;x[89]=0.001;
        x[90]=0.001;x[91]=0.001;x[92]=0.001;x[93]=0.001;
        x[94]=0.001;x[95]=0.001;
        x[96]=0.001;x[97]=0.001;x[98]=0.001;x[99]=0.001;
        x[100]=0.001;x[101]=0.001;x[102]=0.001;


/*      Use BROYDN subroutine from Numerical Recipes book */
/*      by Press et.al */

        broydn(x,N,&check,funcv);

/*      Print the results */

        funcv(N,x,f);
        if (check) printf("Convergence problems.\n");
        printf("%7s %3s %12s\n","Index","x","f");
        for (i=1;i<=N;i++) printf("%5d %12.6f %e\n",i,x[i],f[i]);
        std = 0.0;
        for (i=2;i<=16;i++) std = std + x[i]*x[i];
        for (i=17;i<=25;i++) std = std + 2.0*x[i]*x[i];
        for (i=26;i<=34;i++) std = std + 6.0*x[i]*x[i];
        printf("\nstd.dev. FR = %e\n",sqrt(std));
        std = 0.0;
        for (i=36;i<=50;i++) std = std + x[i]*x[i];
        for (i=51;i<=59;i++) std = std + 2.0*x[i]*x[i];
        for (i=60;i<=68;i++) std = std + 6.0*x[i]*x[i];
        printf("\nstd.dev. delta = %e\n",sqrt(std));
        std = 0.0;
        for (i=70;i<=84;i++) std = std + x[i]*x[i];
        for (i=85;i<=93;i++) std = std + 2.0*x[i]*x[i];
        for (i=94;i<=102;i++) std = std + 6.0*x[i]*x[i];
        printf("\nstd.dev. B = %e\n",sqrt(std));
        free_dvector(f,1,N);
        free_dvector(x,1,N);
        return 0;
}
#undef NRANSI
```

F.2.3 file: aerosolslmodel.c

The following code is an output file in **C** language generated by the prototype language of DEMM. This output file describes the deterministic equivalent model of climate forcing.

```
f[35] =  - al0*fh0*x[69] - al0*fh1*x[74] - 2*al0*fh2*x[89] - 6*al0*fh3 *
x[98] - al1*fh0*x[73] - 2*al2*fh0*x[88] - 6*al3*fh0 *
x[97] + x[35];

f[36] =  - al0*fh0*x[70] + x[36];

f[37] =  - al0*fh0*x[71] - al1*fh0 *
x[80] + x[37];

f[38] =  - al0*fh0*x[72] - al1*fh0*x[82] + x[38];

f[39] =  - al0 *
fh0*x[73] - 2*al1*fh0*x[88] - al1*fh0*x[69] -
al1*fh1*x[74] - 2*al1*fh2*x[89] - 6*al1*fh3*x[98] -
6*al2*fh0*x[97] - 2*al2*fh0*x[73] - 6*al3*fh0*x[88] + x[39];

f[40] =  - al0*fh0 *
x[74] - 2*al0*fh1*x[89] - al0*fh1*x[69] -
6*al0*fh2*x[98] - 2*al0*fh2*x[74] - 6*al0*fh3*x[89] -
al1*fh1*x[73] - 2*al2*fh1*x[88] - 6*al3*fh1*x[97] + x[40];

f[41] =  - al0*fh0 *
x[75] - al1*fh0*x[84] + x[41];

f[42] =  - al0*fh0*x[76] + x[42];

f[43] =  - al0*fh0*x[77] +
x[43];

f[44] =  - al0*fh0*x[78] + x[44];

f[45] =  - al0*fh0*x[79] + x[45];

f[46] =  - al0*fh0*x[80] - al1*fh0 *
x[71] - 2*al2*fh0*x[80] + x[46];

f[47] =  - al0*fh0*x[81] + x[47];

f[48] =  -
al0*fh0*x[82] - al1*fh0*x[72] - 2*al2*fh0*x[82] + x[48];

f[49] =  - al0 *
fh0*x[83] + x[49];

f[50] =  - al0*fh0*x[84] - al1*fh0*x[75] -
2*al2*fh0*x[84] + x[50];

f[51] =  - 2*al0*fh0*x[85] + 2*x[51];

f[52] =  - 2*al0*fh0*x[86] + 2 *
x[52];

f[53] =  - 2*al0*fh0*x[87] + 2*x[53];

f[54] =  - 2*al0*fh0*x[88] -
6*al1*fh0*x[97] - 2*al1*fh0*x[73] - 8*al2*fh0*x[88] - 2*al2*fh0 *
x[69] - 2*al2*fh1*x[74] - 4*al2*fh2*x[89] - 12*al2*fh3 *
x[98] - 36*al3*fh0*x[97] - 6*al3*fh0*x[73] + 2 *
x[54];
```

```
f[55] =  - 2*al0*fh0*x[89] - 6*al0*fh1*x[98] -
2*al0*fh1*x[74] - 8*al0*fh2*x[89] - 2*al0*fh2*x[69] - 36*al0*fh3 *
x[98] - 6*al0*fh3*x[74] - 2*al1*fh2*x[73] - 4*al2*fh2 *
x[88] - 12*al3*fh2*x[97] + 2*x[55];

f[56] =  - 2*al0*fh0*x[90] + 2 *
x[56];

f[57] =  - 2*al0*fh0*x[91] + 2*x[57];

f[58] =  - 2*al0*fh0*x[92] + 2*x[58];

f[59] =  - 2 *
al0*fh0*x[93] + 2*x[59];

f[60] =  - 6*al0*fh0*x[94] + 6*x[60];

f[61] =  - 6*al0*fh0 *
x[95] + 6*x[61];

f[62] =  - 6*al0*fh0*x[96] + 6*x[62];

f[63] =  - 6*al0*fh0*x[97] - 6*al1*fh0 *
x[88] - 36*al2*fh0*x[97] - 6*al2*fh0*x[73] - 36*al3*fh0 *
x[88] - 6*al3*fh0*x[69] - 6*al3*fh1*x[74] - 12*al3*fh2 *
x[89] - 36*al3*fh3*x[98] + 6*x[63];

f[64] =  - 6*al0*fh0*x[98] - 6 *
al0*fh1*x[89] - 36*al0*fh2*x[98] -
6*al0*fh2*x[74] - 36*al0*fh3*x[89] - 6*al0*fh3*x[69] - 6*al1*fh3 *
x[73] - 12*al2*fh3*x[88] - 36*al3*fh3*x[97] + 6 *
x[64];

f[65] =  - 6*al0*fh0*x[99] + 6*x[65];

f[66] =  - 6*al0*fh0*x[100] + 6*x[66];

f[67] =  -
6*al0*fh0*x[101] + 6*x[67];

f[68] =  - 6*al0*fh0*x[102] + 6*x[68];

f[69] = pow( - A,1.0*(-1))*Y0 *
Q0*3.0*ts0 + x[69];

f[70] = pow( - A,1.0*(-1))*Y0*Q1*3.0*ts0 + x[70];

f[71] = pow( - A,1.0*(-1))*Y1*Q0*3.0*ts0 +
x[71];

f[72] = pow( - A,1.0*(-1))*Y0*Q0*3.0*ts1 + x[72];

f[73] = x[73];

f[74] = x[74];

f[75] = x[75];
```

```
f[76] = x[76];

f[77] = x[77];

f[78] = x[78];

f[79] = pow( - A,1.0*(-1))*Y1*Q0 *
3.0*ts1 + x[79];

f[80] = x[80];

f[81] = x[81];

f[82] = x[82];

f[83] = x[83];

f[84] = x[84];

f[85] =  - 2*pow(A,1.0*(-1))*Y0*Q2*3.0*ts0 +
2*x[85];

f[86] =  - 2*pow(A,1.0*(-1))*Y2*Q0*3.0*ts0 + 2*x[86];

f[87] =  - 2*pow(A,1.0*(-1))*Y0*Q0*3.0 *
ts2 + 2*x[87];

f[88] = 2*x[88];

f[89] = 2*x[89];

f[90] = 2*x[90];

f[91] = 2*x[91];

f[92] = 2*x[92];

f[93] = 2*x[93];

f[94] =  - 6*
pow(A,1.0*(-1))*Y0*Q3*3.0*ts0 + 6*x[94];

f[95] =  - 6*pow(A,1.0*(-1))*Y3*Q0*3.0*ts0 + 6 *
x[95];

f[96] =  - 6*pow(A,1.0*(-1))*Y0*Q0*3.0*ts3 + 6*x[96];

f[97] = 6*x[97];

f[98] = 6*x[98];

f[99] = 6 *
x[99];

f[100] = 6*x[100];

f[101] = 6*x[101];

f[102] = 6*x[102];
```

```
f[1] = 2*T0*0.5*ft*b0*ma2*ms0*x[57] + 6*T0*0.5*ft*b0 *
ma3*ms0*x[66] +
2*T0*0.5*ft*b0*ma0*ms2*x[58] + 6*T0*0.5*ft*b0*ma0*ms3*x[67] + T0*0.5*ft*b0*ma0 *
ms0*x[35] + T0*0.5*ft*b0*ma0*ms1*x[43] + T0*0.5 *
ft*b0*ma1*ms0*x[42] +
T0*0.5*ft*b1*ma0*ms0*x[44] + 2*T0*0.5*ft*b2*ma0*ms0*x[59] + 6*T0*0.5*ft*b3*ma0 *
ms0*x[68] + T1*0.5*ft*b0*ma0*ms0*x[41] + 2*T2 *
0.5*ft*b0*ma0*ms0*x[56] +
6*T3*0.5*ft*b0*ma0*ms0*x[65] + x[1];

f[2] = T0*0.5*ft*b0*ma0*ms0*x[36] + x[2];

f[3] = T0*0.5*ft*b0*ma0*ms0 *
x[37] + T1*0.5*ft*b0*ma0*ms0*x[47] + x[3];

f[4] = T0*0.5*ft*b0*ma0*ms0 *
x[38] + T1*0.5*ft*b0*ma0*ms0*x[49] + x[4];

f[5] = T0*0.5*ft*b0*ma0*ms0 *
x[39] + T1*0.5*ft*b0*ma0*ms0*x[50] + x[5];

f[6] = T0*0.5*ft*b0*ma0*ms0 *
x[40] + x[6];

f[7] = T0*0.5*ft*b0*ma0*ms0*x[41] +
2*T1*0.5*ft*b0*ma2*ms0*x[57] + 6*T1*0.5*ft*b0*ma3*ms0*x[66] + 2*T1*0.5*ft*b0 *
ma0*ms2*x[58] +
6*T1*0.5*ft*b0*ma0*ms3*x[67] + T1*0.5*ft*b0*ma0*ms0*x[35] + 2*T1*0.5*ft*b0*ma0 *
ms0*x[56] + T1*0.5*ft*b0*ma0*ms1*x[43] + T1 *
0.5*ft*b0*ma1*ms0*x[42] +
T1*0.5*ft*b1*ma0*ms0*x[44] + 2*T1*0.5*ft*b2*ma0*ms0*x[59] + 6*T1*0.5*ft*b3*ma0 *
ms0*x[68] + 6*T2*0.5*ft*b0*ma0*ms0*x[65] + 2*T2 *
0.5*ft*b0*ma0*ms0*x[41] +
6*T3*0.5*ft*b0*ma0*ms0*x[56] + x[7];

f[8] = 6*T0*0.5*ft*b0*ma2*ms0*x[66] + 2*T0*0.5*ft*b0*ma2*ms0 *
x[42] + 6*T0*0.5*ft*b0*ma3*ms0*x[57] + T0*0.5*ft *
b0*ma0*ms0*x[42] +
2*T0*0.5*ft*b0*ma1*ms2*x[58] + 6*T0*0.5*ft*b0*ma1*ms3*x[67] + T0*0.5*ft*b0*ma1 *
ms0*x[35] + 2*T0*0.5*ft*b0*ma1*ms0*x[57] + T0 *
0.5*ft*b0*ma1*ms1*x[43] +
T0*0.5*ft*b1*ma1*ms0*x[44] + 2*T0*0.5*ft*b2*ma1*ms0*x[59] + 6*T0*0.5*ft*b3*ma1 *
ms0*x[68] + T1*0.5*ft*b0*ma1*ms0*x[41] + 2*T2 *
0.5*ft*b0*ma1*ms0*x[56] +
6*T3*0.5*ft*b0*ma1*ms0*x[65] + x[8];

f[9] = 2*T0*0.5*ft*b0*ma2*ms1*x[57] + 6*T0*0.5*ft*b0*ma3*ms1 *
x[66] + 6*T0*0.5*ft*b0*ma0*ms2*x[67] + 2*T0*0.5 *
ft*b0*ma0*ms2*x[43] +
6*T0*0.5*ft*b0*ma0*ms3*x[58] + T0*0.5*ft*b0*ma0*ms0*x[43] + 2*T0*0.5*ft*b0*ma0 *
ms1*x[58] + T0*0.5*ft*b0*ma0*ms1*x[35] + T0 *
0.5*ft*b0*ma1*ms1*x[42] +
T0*0.5*ft*b1*ma0*ms1*x[44] + 2*T0*0.5*ft*b2*ma0*ms1*x[59] + 6*T0*0.5*ft*b3*ma0 *
ms1*x[68] + T1*0.5*ft*b0*ma0*ms1*x[41] + 2*T2 *
0.5*ft*b0*ma0*ms1*x[56] +
6*T3*0.5*ft*b0*ma0*ms1*x[65] + x[9];

f[10] = T0*0.5*ft*b0*ma0*ms0*x[44] + 2*T0*0.5*ft*b1*ma2*ms0 *
```

```
x[57] + 6*T0*0.5*ft*b1*ma3*ms0*x[66] + 2*T0*0.5 *
ft*b1*ma0*ms2*x[58] +
6*T0*0.5*ft*b1*ma0*ms3*x[67] + 2*T0*0.5*ft*b1*ma0*ms0*x[59] + T0*0.5*ft*b1*ma0 *
ms0*x[35] + T0*0.5*ft*b1*ma0*ms1*x[43] + T0*0.5 *
ft*b1*ma1*ms0*x[42] +
6*T0*0.5*ft*b2*ma0*ms0*x[68] + 2*T0*0.5*ft*b2*ma0*ms0*x[44] + 6*T0*0.5*ft*b3 *
ma0*ms0*x[59] + T1*0.5*ft*b1*ma0*ms0*x[41] + 2 *
T2*0.5*ft*b1*ma0*ms0*x[56] +
6*T3*0.5*ft*b1*ma0*ms0*x[65] + x[10];

f[11] = T0*0.5*ft*b0*ma0*ms0*x[45] + x[11];

f[12] = T0*0.5*ft*b0*ma0 *
ms0*x[46] + x[12];

f[13] = T0*0.5*ft*b0*ma0*ms0*x[47] +
T1*0.5*ft*b0*ma0*ms0*x[37] + 2*T2*0.5*ft*b0*ma0*ms0*x[47] + x[13];

f[14] = T0*0.5 *
ft*b0*ma0*ms0*x[48] + x[14];

f[15] = T0*0.5*ft*b0*ma0*ms0*x[49] + T1*0.5*ft*b0*ma0*ms0 *
x[38] + 2*T2*0.5*ft*b0*ma0*ms0*x[49] + x[15];

f[16] = T0 *
0.5*ft*b0*ma0*ms0*x[50] + T1*0.5*ft*b0*ma0*ms0*x[39] + 2*T2*0.5*ft*b0 *
ma0*ms0*x[50] + x[16];

f[17] = 2*T0*0.5*ft*b0*ma0*ms0*x[51] + 2 *
x[17];

f[18] = 2*T0*0.5*ft*b0*ma0*ms0*x[52] + 2*x[18];

f[19] = 2*T0*0.5*ft*b0*ma0*ms0 *
x[53] + 2*x[19];

f[20] = 2*T0*0.5*ft*b0*ma0*ms0*x[54] + 2*x[20];

f[21] = 2*T0*0.5*ft*b0 *
ma0*ms0*x[55] + 2*x[21];

f[22] = 2*T0*0.5*ft*b0*ma0*ms0*x[56] + 6*T1*0.5*ft*b0*ma0*ms0 *
x[65] + 2*T1*0.5*ft*b0*ma0*ms0*x[41] + 4*T2*0.5 *
ft*b0*ma2*ms0*x[57] +
12*T2*0.5*ft*b0*ma3*ms0*x[66] + 4*T2*0.5*ft*b0*ma0*ms2*x[58] + 12*T2*0.5*ft*b0 *
ma0*ms3*x[67] +
2*T2*0.5*ft*b0*ma0*ms0*x[35] + 8*T2*0.5*ft*b0*ma0*ms0*x[56] + 2*T2*0.5*ft*b0 *
ma0*ms1*x[43] +
2*T2*0.5*ft*b0*ma1*ms0*x[42] + 2*T2*0.5*ft*b1*ma0*ms0*x[44] + 4*T2*0.5*ft*b2 *
ma0*ms0*x[59] +
12*T2*0.5*ft*b3*ma0*ms0*x[68] + 36*T3*0.5*ft*b0*ma0*ms0*x[65] + 6*T3*0.5*ft*b0 *
ma0*ms0*x[41] + 2*x[22];

f[23] = 4*T0*0.5*ft*b0*ma2*ms2*x[58] + 12 *
T0*0.5*ft*b0*ma2*ms3*x[67] +
2*T0*0.5*ft*b0*ma2*ms0*x[35] + 8*T0*0.5*ft*b0*ma2*ms0*x[57] + 2*T0*0.5*ft*b0 *
ma2*ms1*x[43] +
36*T0*0.5*ft*b0*ma3*ms0*x[66] + 6*T0*0.5*ft*b0*ma3*ms0*x[42] + 2*T0*0.5*ft*b0 *
ma0*ms0*x[57] +
```

```
6*T0*0.5*ft*b0*ma1*ms0*x[66] + 2*T0*0.5*ft*b0*ma1*ms0*x[42] + 2*T0*0.5*ft*b1 *
ma2*ms0*x[44] +
4*T0*0.5*ft*b2*ma2*ms0*x[59] + 12*T0*0.5*ft*b3*ma2*ms0*x[68] + 2*T1*0.5*ft*b0 *
ma2*ms0*x[41] +
4*T2*0.5*ft*b0*ma2*ms0*x[56] + 12*T3*0.5*ft*b0*ma2*ms0*x[65] +
2*x[23];

f[24] = 4*T0*0.5*ft*b0*ma2*ms2*x[57] +
12*T0*0.5*ft*b0*ma3*ms2*x[66] + 8*T0*0.5*ft*b0*ma0*ms2*x[58] + 2*T0*0.5*ft*b0 *
ma0*ms2*x[35] +
36*T0*0.5*ft*b0*ma0*ms3*x[67] + 6*T0*0.5*ft*b0*ma0*ms3*x[43] + 2*T0*0.5*ft*b0 *
ma0*ms0*x[58] +
6*T0*0.5*ft*b0*ma0*ms1*x[67] + 2*T0*0.5*ft*b0*ma0*ms1*x[43] + 2*T0*0.5*ft*b0 *
ma1*ms2*x[42] +
2*T0*0.5*ft*b1*ma0*ms2*x[44] + 4*T0*0.5*ft*b2*ma0*ms2*x[59] + 12*T0*0.5*ft*b3 *
ma0*ms2*x[68] +
2*T1*0.5*ft*b0*ma0*ms2*x[41] + 4*T2*0.5*ft*b0*ma0*ms2*x[56] + 12*T3*0.5*ft*b0 *
ma0*ms2*x[65] + 2*x[24];

f[25] = 2*T0*0.5*ft*b0*ma0*ms0*x[59] + 6 *
T0*0.5*ft*b1*ma0*ms0*x[68] +
2*T0*0.5*ft*b1*ma0*ms0*x[44] + 4*T0*0.5*ft*b2*ma2*ms0*x[57] + 12*T0*0.5*ft*b2 *
ma3*ms0*x[66] +
4*T0*0.5*ft*b2*ma0*ms2*x[58] + 12*T0*0.5*ft*b2*ma0*ms3*x[67] + 8*T0*0.5*ft*b2 *
ma0*ms0*x[59] +
2*T0*0.5*ft*b2*ma0*ms0*x[35] + 2*T0*0.5*ft*b2*ma0*ms1*x[43] + 2*T0*0.5*ft*b2 *
ma1*ms0*x[42] +
36*T0*0.5*ft*b3*ma0*ms0*x[68] + 6*T0*0.5*ft*b3*ma0*ms0*x[44] + 2*T1*0.5*ft*b2 *
ma0*ms0*x[41] +
4*T2*0.5*ft*b2*ma0*ms0*x[56] + 12*T3*0.5*ft*b2*ma0*ms0*x[65] +
2*x[25];

f[26] = 6*T0*0.5*ft*b0*ma0*ms0*x[60] + 6*x[26];

f[27] = 6*T0*0.5*ft*b0*ma0*ms0*x[61] + 6 *
x[27];

f[28] = 6*T0*0.5*ft*b0*ma0*ms0*x[62] + 6*x[28];

f[29] = 6*T0*0.5*ft*b0*ma0*ms0 *
x[63] + 6*x[29];

f[30] = 6*T0*0.5*ft*b0*ma0*ms0*x[64] + 6*x[30];

f[31] = 6*T0*0.5*ft*b0 *
ma0*ms0*x[65] + 6*T1*0.5*ft*b0*ma0*ms0*x[56] + 36*T2*0.5*ft*b0*ma0 *
ms0*x[65] + 6*T2*0.5*ft*b0*ma0*ms0*x[41] + 12 *
T3*0.5*ft*b0*ma2*ms0*x[57] +
36*T3*0.5*ft*b0*ma3*ms0*x[66] + 12*T3*0.5*ft*b0*ma0*ms2*x[58] + 36*T3*0.5*ft*b0 *
ma0*ms3*x[67] +
6*T3*0.5*ft*b0*ma0*ms0*x[35] + 36*T3*0.5*ft*b0*ma0*ms0*x[56] + 6*T3*0.5*ft*b0 *
ma0*ms1*x[43] +
6*T3*0.5*ft*b0*ma1*ms0*x[42] + 6*T3*0.5*ft*b1*ma0*ms0*x[44] + 12*T3*0.5*ft*b2 *
ma0*ms0*x[59] +
36*T3*0.5*ft*b3*ma0*ms0*x[68] + 6*x[31];

f[32] = 36*T0*0.5*ft*b0*ma2*ms0*x[66] + 6*T0*0.5*ft*b0*ma2 *
ms0*x[42] + 12*T0*0.5*ft*b0*ma3*ms2*x[58] + 36 *
T0*0.5*ft*b0*ma3*ms3*x[67] +
```

```
6*T0*0.5*ft*b0*ma3*ms0*x[35] + 36*T0*0.5*ft*b0*ma3*ms0*x[57] + 6*T0*0.5*ft*b0 *
ma3*ms1*x[43] +
6*T0*0.5*ft*b0*ma0*ms0*x[66] + 6*T0*0.5*ft*b0*ma1*ms0*x[57] + 6*T0*0.5*ft*b1 *
ma3*ms0*x[44] +
12*T0*0.5*ft*b2*ma3*ms0*x[59] + 36*T0*0.5*ft*b3*ma3*ms0*x[68] + 6*T1*0.5*ft*b0 *
ma3*ms0*x[41] +
12*T2*0.5*ft*b0*ma3*ms0*x[56] + 36*T3*0.5*ft*b0*ma3*ms0*x[65] +
6*x[32];

f[33] = 12*T0*0.5*ft*b0*ma2*ms3*x[57] +
36*T0*0.5*ft*b0*ma3*ms3*x[66] + 36*T0*0.5*ft*b0*ma0*ms2*x[67] + 6*T0*0.5*ft*b0 *
ma0*ms2*x[43] +
36*T0*0.5*ft*b0*ma0*ms3*x[58] + 6*T0*0.5*ft*b0*ma0*ms3*x[35] + 6*T0*0.5*ft*b0 *
ma0*ms0*x[67] +
6*T0*0.5*ft*b0*ma0*ms1*x[58] + 6*T0*0.5*ft*b0*ma1*ms3*x[42] + 6*T0*0.5*ft*b1 *
ma0*ms3*x[44] +
12*T0*0.5*ft*b2*ma0*ms3*x[59] + 36*T0*0.5*ft*b3*ma0*ms3*x[68] + 6*T1*0.5*ft*b0 *
ma0*ms3*x[41] +
12*T2*0.5*ft*b0*ma0*ms3*x[56] + 36*T3*0.5*ft*b0*ma0*ms3*x[65] +
6*x[33];

f[34] = 6*T0*0.5*ft*b0*ma0*ms0*x[68] +
6*T0*0.5*ft*b1*ma0*ms0*x[59] + 36*T0*0.5*ft*b2*ma0*ms0*x[68] + 6*T0*0.5*ft*b2 *
ma0*ms0*x[44] +
12*T0*0.5*ft*b3*ma2*ms0*x[57] + 36*T0*0.5*ft*b3*ma3*ms0*x[66] + 12*T0*0.5*ft*b3 *
ma0*ms2*x[58] +
36*T0*0.5*ft*b3*ma0*ms3*x[67] + 36*T0*0.5*ft*b3*ma0*ms0*x[59] + 6*T0*0.5*ft*b3 *
ma0*ms0*x[35] +
6*T0*0.5*ft*b3*ma0*ms1*x[43] + 6*T0*0.5*ft*b3*ma1*ms0*x[42] + 6*T1*0.5*ft*b3 *
ma0*ms0*x[41] +
12*T2*0.5*ft*b3*ma0*ms0*x[56] + 36*T3*0.5*ft*b3*ma0*ms0*x[65] +
6*x[34];
```

F.2.4 file: aerosolsl.res

The following text is the result file from running the output files generated by the prototype language of DEMM. This result file reports the values of coefficients of response variables

Index	x	f
1	-1.326356	8.881784e-16
2	0.173334	-5.551115e-17
3	-0.535419	-1.110223e-16
4	-0.535420	-1.110223e-16
5	-0.458905	-2.220446e-16
6	-0.243518	5.551115e-17
7	0.381221	1.165734e-15
8	-0.126677	-1.360023e-15
9	0.223389	-1.026956e-15
10	0.311071	-1.665335e-16
11	-0.216136	-5.551115e-17
12	-0.185249	-8.326673e-17
13	0.153890	-8.326673e-17
14	-0.185249	-1.110223e-16
15	0.153890	-8.326673e-17

16	0.131898	4.996004e-16
17	-0.027419	-1.457168e-16
18	-0.117640	-1.387779e-16
19	-0.117641	-1.110223e-16
20	-0.036366	-9.714451e-16
21	-0.008950	-8.257284e-16
22	-0.158980	-6.272760e-15
23	-0.001813	-4.767887e-15
24	-0.046597	-4.843348e-15
25	-0.096371	9.992007e-16
26	-0.022925	-5.828671e-16
27	0.000000	-5.692465e-16
28	0.000000	-5.692465e-16
29	-0.008573	-2.289835e-16
30	0.000000	-5.928476e-16
31	-0.050003	2.531308e-14
32	-0.000000	-9.765173e-16
33	-0.029486	3.608225e-15
34	-0.040869	-2.337019e-14
35	0.035424	-9.020562e-17
36	-0.004629	-1.734723e-18
37	0.014300	1.734723e-18
38	0.014300	1.734723e-18
39	0.012256	-6.938894e-18
40	0.006504	1.040834e-17
41	0.000000	1.372643e-19
42	-0.000000	-1.336236e-18
43	-0.000000	-1.336236e-18
44	-0.000000	-1.336236e-18
45	0.005773	-8.673617e-19
46	0.004948	-4.336809e-18
47	-0.000000	-1.336236e-18
48	0.004948	-4.336809e-18
49	-0.000000	-1.336236e-18
50	-0.000000	-2.428995e-18
51	0.000732	-2.385245e-18
52	0.003142	-1.734723e-18
53	0.003142	-1.734723e-18
54	0.000971	-7.177418e-17
55	0.000239	-5.946849e-17
56	-0.000000	-2.672472e-18
57	-0.000000	-2.672472e-18
58	-0.000000	-2.672472e-18
59	-0.000000	-2.672472e-18
60	0.000612	-1.561251e-17
61	-0.000000	-1.629168e-17
62	-0.000000	-1.629168e-17
63	0.000229	9.909608e-17

```
  64    -0.000000 -2.426107e-17
  65    -0.000000 -1.629168e-17
  66    -0.000000 -1.629168e-17
  67    -0.000000 -1.629168e-17
  68    -0.000000 -1.629168e-17
  69     0.003873 0.000000e+00
  70    -0.000506 0.000000e+00
  71     0.001564 0.000000e+00
  72     0.001564 0.000000e+00
  73     0.000000 0.000000e+00
  74     0.000000 0.000000e+00
  75     0.000000 0.000000e+00
  76     0.000000 0.000000e+00
  77     0.000000 0.000000e+00
  78     0.000000 0.000000e+00
  79     0.000631 0.000000e+00
  80     0.000000 0.000000e+00
  81     0.000000 0.000000e+00
  82     0.000000 0.000000e+00
  83     0.000000 0.000000e+00
  84     0.000000 0.000000e+00
  85     0.000080 0.000000e+00
  86     0.000344 0.000000e+00
  87     0.000344 0.000000e+00
  88     0.000000 0.000000e+00
  89     0.000000 0.000000e+00
  90     0.000000 0.000000e+00
  91     0.000000 0.000000e+00
  92     0.000000 0.000000e+00
  93     0.000000 0.000000e+00
  94     0.000067 5.421011e-20
  95     0.000000 0.000000e+00
  96     0.000000 0.000000e+00
  97     0.000000 0.000000e+00
  98     0.000000 0.000000e+00
  99     0.000000 0.000000e+00
 100     0.000000 0.000000e+00
 101     0.000000 0.000000e+00
 102     0.000000 0.000000e+00

std.dev. FR = 1.236291e+00

std.dev. delta = 2.739225e-02

std.dev. B = 2.460856e-03
```

F.3 Small gas-phase process flowsheet

F.3.1 file: StochasticPlantEx.dem

The following code is an input file written in the prototype language of DEMM. This input file describes the stochastic model of gas phase process flowsheet that is represented by a set of algebraic equations in the **Abacuss** process simulator environment.

```
%{

#-------------------------------------------------------------#
# example for stochastic steady state plant model             #
#-------------------------------------------------------------#
#                                                             #
# taken from Naf, U.G., "Stochastic Simulation using gPROMS"  #
# Technical Report, Centre for Process System Engineering     #
# Imperial College (1992).                                    #
#                                                             #
# created         : Menner A. Tatang / 4.28.94                #
# last modified   : Menner A. Tatang / 5.02.94                #
#                                                             #
#-------------------------------------------------------------#

DECLARE

TYPE
  Molar_Flow      =   250   :    -10000    :   10000  UNIT="mol/s"
  Temperature     =   300   :    -1000     :   1000   UNIT="K"
  Pressure        =   10.   :    -50       :   50     UNIT="bara"
  Enthalpy        =   1.0   :    -1e4      :   1e4    UNIT="kJ/mol"
  Mole_Fraction   =   0.33  :    -100.01   :   100.01
  M_Fraction      =   0.33  :    -100      :   100.01
  Notype          =   0.0   :    -1e-4     :   1e-4
  NotypeO         =   1     :    -1e5      :   1e5
  Fraction        =   0.5   :    -100.01   :   100.01

STREAM
  MainStream    IS Molar_Flow,Temperature,Pressure,Mole_Fraction

END

#-------------------------------------------------------------

%}

%Format ABACUSS

%%

# define common basis, moments values, and polynomial chaos

Set [BF] basis = {e1,e2,e3,e4};

Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135,0,2027025};

Set [PC] pcall = {1,e1,e2,e3,e4,e1^2-1,e2^2-1,e3^2-1,e4^2-1};

%{
```

```
MODEL Stochastic_Mixer

PARAMETER
  NO_COMP       AS INTEGER
  NO_STREAMS    AS INTEGER
  NO_PC         AS INTEGER
  AP            AS REAL
  BP            AS REAL
  CP            AS REAL

VARIABLE
  Flow_In       AS ARRAY(NO_STREAMS,NO_PC) OF Molar_Flow
  Flow_Out      AS ARRAY(NO_PC)            OF Molar_Flow
  P_In          AS ARRAY(NO_STREAMS) OF Pressure
  P_Out         AS Pressure
  T_In          AS ARRAY(NO_STREAMS,NO_PC) OF Temperature
  T_Out         AS ARRAY(NO_PC)            OF Temperature
  X_In          AS ARRAY(NO_STREAMS,NO_COMP,NO_PC) OF Mole_Fraction
  X_Out         AS ARRAY(NO_COMP,NO_PC)            OF Mole_Fraction
  A             AS ARRAY(NO_COMP,NO_PC) OF Notype
  B             AS ARRAY(NO_PC) OF Notype
  C             AS ARRAY(NO_PC) OF Notype
  AD            AS ARRAY(NO_COMP) OF M_Fraction

STREAM
  Inlets : Flow_In ,T_In ,P_In ,X_In  AS ARRAY(NO_STREAMS) OF MainStream
  Outlet : Flow_Out,T_Out,P_Out,X_Out AS MainStream

SET
  AP    := 0.0;
  BP    := 0.0;
  CP    := 0.0;

EQUATION
  FOR I:=1 TO NO_COMP DO

%}

Start mixermodel1

# put local random parameters and variables

Set [RP] X_In[1][I] -> pcall = {{X_In[1][I][1,...,9]}},
         X_In[2][I] -> pcall = {{X_In[2][I][1,...,9]}},
         Flow_In[1] -> pcall = {{Flow_In[1][1,...,9]}},
         Flow_In[2] -> pcall = {{Flow_In[2][1,...,9]}};

Set [RV] X_Out[I] -> pcall = {{X_Out[I][1,...,9]}},
         Flow_Out -> pcall = {{Flow_Out[1,...,9]}};

Models eq1 := Flow_Out * X_Out[I] -
              Flow_In[1] * X_In[1][I] - Flow_In[2] * X_In[2][I]
            = {{A[I][1,...,9]} -> {pcall[1,...,9]}};

End

%{
```

```
   END # for

   FOR I:=1 TO NO_COMP DO
   FOR J:=1 TO NO_PC DO
       A(I,J) = AP;
   END # for j
   END # for i

%}

Start mixermodel2

# put local random parameters and variables

Set [RP] T_In[1] -> pcall = {{T_In[1][1,...,9]}},
         T_In[2] -> pcall = {{T_In[2][1,...,9]}},
         Flow_In[1] -> pcall = {{Flow_In[1][1,...,9]}},
         Flow_In[2] -> pcall = {{Flow_In[2][1,...,9]}};

Set [RV] X_Out[1] -> pcall = {{X_Out[1][1,...,9]}},
         X_Out[2] -> pcall = {{X_Out[2][1,...,9]}},
         X_Out[3] -> pcall = {{X_Out[3][1,...,9]}},
         T_Out -> pcall = {{T_Out[1,...,9]}},
         Flow_Out -> pcall = {{Flow_Out[1,...,9]}};

Models eq1 := X_Out[1] + X_Out[2] + X_Out[3] - 1.0
           = {{B[1,...,9]} -> {pcall[1,...,9]}},
       eq2 := Flow_Out * T_Out -
              Flow_In[1] * T_In[1] - Flow_In[2] * T_In[2]
           = {{C[1,...,9]} -> {pcall[1,...,9]}};

End

%{

   FOR I:=1 TO NO_PC DO
       B(I) = BP;
       C(I) = CP;
   END # for

   P_Out = MIN( P_In );

   FOR I:=1 TO NO_COMP DO
       AD(I) = X_Out(I,1);
   END # for

 END    # Stochastic Mixer
 #.....................................................

 MODEL Stochastic_Splitter

PARAMETER
  NO_STREAMS      AS INTEGER
  NO_COMP         AS INTEGER
  NO_PC           AS INTEGER
  AP              AS REAL
  BP              AS REAL
  CP              AS REAL
  DP              AS REAL
```

```
 EP              AS REAL
 FP              AS REAL

VARIABLE
  Flow_Out       AS ARRAY(NO_STREAMS,NO_PC)   OF Molar_Flow
  Flow_In        AS ARRAY(NO_PC)              OF Molar_Flow
  P_Out          AS ARRAY(NO_STREAMS)   OF Pressure
  P_In           AS                          Pressure
  T_Out          AS ARRAY(NO_STREAMS,NO_PC)   OF Temperature
  T_In           AS ARRAY(NO_PC)              OF Temperature
  X_In           AS ARRAY(NO_COMP,NO_PC)      OF Mole_Fraction
  X_O            AS ARRAY(NO_STREAMS,NO_COMP,NO_PC) OF Mole_Fraction
  Fraction_To_1  AS                          Fraction
  A             AS ARRAY(NO_PC) OF Notype
  B             AS ARRAY(NO_PC) OF Notype
  C             AS ARRAY(NO_PC) OF Notype
  D             AS ARRAY(NO_PC) OF Notype
  E             AS ARRAY(NO_PC) OF Notype
  F             AS ARRAY(NO_PC) OF Notype
  AD            AS ARRAY(NO_COMP) OF M_Fraction
  BD            AS ARRAY(NO_COMP) OF M_Fraction

STREAM
  Inlet   : Flow_In ,T_In ,P_In ,X_In  AS                      MainStream
  Outlets : Flow_Out,T_Out,P_Out,X_O   AS ARRAY(NO_STREAMS) OF MainStream

SET
  AP  := 0.0;
  BP  := 0.0;
  CP  := 0.0;
  DP  := 0.0;
  EP  := 0.0;
  FP  := 0.0;

EQUATION

%}

Start splittermodel1

# put local random parameters and variables

Set [RP] Flow_In -> pcall = {{Flow_In[1,...,9]}},
         T_In -> pcall = {{T_In[1,...,9]}};

Set [RV] Flow_Out[1] -> pcall = {{Flow_Out[1][1,...,9]}},
         Flow_Out[2] -> pcall = {{Flow_Out[2][1,...,9]}},
         T_Out[1] -> pcall = {{T_Out[1][1,...,9]}},
         T_Out[2] -> pcall = {{T_Out[2][1,...,9]}};

Models eq1 := Flow_Out[1] - Fraction_To_1 * Flow_In
           = {{A[1,...,9]} -> {pcall[1,...,9]}},
       eq2 := Flow_In - Flow_Out[1] - Flow_Out[2]
           = {{B[1,...,9]} -> {pcall[1,...,9]}},
       eq3 := T_Out[1] - T_In
           = {{C[1,...,9]} -> {pcall[1,...,9]}},
       eq4 := T_Out[2] - T_In
           = {{D[1,...,9]} -> {pcall[1,...,9]}};
```

```
End

%{

   FOR I:=1 TO NO_PC DO
       A(I) = AP;
       B(I) = BP;
       C(I) = CP;
       D(I) = DP;
   END # for

   P_Out =  P_In;

   FOR I:= 1 TO NO_COMP DO

%}

Start splittermodel2

# put local random parameters and variables

Set [RP] X_In[I] -> pcall = {{X_In[I][1,...,9]}};

Set [RV] X_O[1][I] -> pcall = {{X_O[1][I][1,...,9]}},
         X_O[2][I] -> pcall = {{X_O[2][I][1,...,9]}};

Models eq1 := X_O[1][I] - X_In[I]
           = {{E[1,...,9]} -> {pcall[1,...,9]}},
       eq2 := X_O[2][I] - X_In[I]
           = {{F[1,...,9]} -> {pcall[1,...,9]}};

End

%{

   END # for

   FOR I:=1 TO NO_PC DO
       E(I) = EP;
       F(I) = FP;
   END # for

   FOR I:=1 TO NO_COMP DO
       AD(I) = X_O(1,I,1);
       BD(I) = X_O(2,I,1);
   END # for

END    # Stochastic Splitter
#.................................................

MODEL Stochastic_Flash

PARAMETER
  NO_COMP        AS INTEGER
  NO_PC          AS INTEGER
  AP             AS REAL
  BP             AS REAL
  CP             AS REAL
  DP             AS REAL
```

```
  EP              AS REAL
  FP              AS REAL

 VARIABLE
  Flow_Vap        AS ARRAY(NO_PC)          OF Molar_Flow
  X_Vap           AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
  P_Vap           AS                       Pressure
  T_Vap           AS ARRAY(NO_PC)          OF Temperature
  Flow_Liq        AS ARRAY(NO_PC)          OF Molar_Flow
  P_Liq           AS                       Pressure
  X_Liq           AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
  T_Liq           AS ARRAY(NO_PC)          OF Temperature
  Flow_In         AS ARRAY(NO_PC)          OF Molar_Flow
  P_In            AS                       Pressure
  T_In            AS ARRAY(NO_PC)          OF Temperature
  X_In            AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
  P_Flash         AS                       Pressure
  T_Flash         AS ARRAY(NO_PC)          OF Temperature
  K               AS ARRAY(NO_COMP) OF NotypeO
  A               AS ARRAY(NO_COMP,NO_PC) OF Notype
  B               AS ARRAY(NO_PC)          OF Notype
  C               AS ARRAY(NO_PC)          OF Notype
  D               AS ARRAY(NO_COMP,NO_PC) OF Notype
  E               AS ARRAY(NO_PC)          OF Notype
  F               AS ARRAY(NO_PC)          OF Notype
  AD              AS ARRAY(NO_COMP) OF M_Fraction
  BD              AS ARRAY(NO_COMP) OF M_Fraction

 STREAM
  Inlet   : Flow_In ,T_In ,P_In ,X_In  AS MainStream
  Liquid  : Flow_Liq,T_Liq,P_Liq,X_Liq AS MainStream
  Vapour  : Flow_Vap,T_Vap,P_Vap,X_Vap AS MainStream

 SET
  AP  := 0.0;
  BP  := 0.0;
  CP  := 0.0;
  DP  := 0.0;
  EP  := 0.0;
  FP  := 0.0;

 EQUATION

  FOR I:=1 TO NO_COMP DO

%}

Start flashmodel1

# put local random parameters and variables

Set [RV] X_Vap[I] -> pcall = {{X_Vap[I][1,...,9]}},
         X_Liq[I] -> pcall = {{X_Liq[I][1,...,9]}};

Models eq1 := X_Vap[I] - X_Liq[I] * K[I]
           = {{A[I][1,...,9]} -> {pcall[1,...,9]}};

End
```

```
%{

   END # for

   FOR I:=1 TO NO_COMP DO
   FOR J:=1 TO NO_PC DO
       A(I,J) = AP;
   END # for j
   END # for i

%}

Start flashmodel2

# put local random parameters and variables

Set [RV] X_Vap[1] -> pcall = {{X_Vap[1][1,...,9]}},
         X_Vap[2] -> pcall = {{X_Vap[2][1,...,9]}},
         X_Vap[3] -> pcall = {{X_Vap[3][1,...,9]}},
         X_Liq[1] -> pcall = {{X_Liq[1][1,...,9]}},
         X_Liq[2] -> pcall = {{X_Liq[2][1,...,9]}},
         X_Liq[3] -> pcall = {{X_Liq[3][1,...,9]}};

Models eq1 := X_Vap[1] + X_Vap[2] + X_Vap[3] - 1.0
           = {{B[1,...,9]} -> {pcall[1,...,9]}},
       eq2 := X_Liq[1] + X_Liq[2] + X_Liq[3] - 1.0
           = {{C[1,...,9]} -> {pcall[1,...,9]}};

End

%{

   FOR I:=1 TO NO_PC DO
       B(I) = BP;
       C(I) = CP;
   END # for

   FOR I:=1 TO NO_COMP DO

%}

Start flashmodel3

Set [RP] X_Vap[I] -> pcall = {{X_Vap[I][1,...,9]}},
         X_Liq[I] -> pcall = {{X_Liq[I][1,...,9]}},
         X_In[I] -> pcall = {{X_In[I][1,...,9]}},
         Flow_In -> pcall = {{Flow_In[1,...,9]}};

Set [RV] Flow_Vap -> pcall = {{Flow_Vap[1,...,9]}},
         Flow_Liq -> pcall = {{Flow_Liq[1,...,9]}};

Models eq1 := Flow_In * X_In[I] - Flow_Vap * X_Vap[I] - Flow_Liq * X_Liq[I]
           = {{D[I][1,...,9]} -> {pcall[1,...,9]}};

End

%{

   END # for
```

```
   FOR I:=1 TO NO_COMP DO
   FOR J:=1 TO NO_PC DO
       D(I,J) = DP;
   END # for j
   END # for i
%}

Start flashmodel4

Set [RP] T_Flash -> pcall = {{T_Flash[1,...,9]}};

Set [RV] T_Liq -> pcall = {{T_Liq[1,...,9]}},
         T_Vap -> pcall = {{T_Vap[1,...,9]}};

Models eq1 := T_Liq - T_Flash
           = {{E[1,...,9]} -> {pcall[1,...,9]}},
       eq2 := T_Vap - T_Flash
           = {{F[1,...,9]} -> {pcall[1,...,9]}};

End

%{

   FOR I:=1 TO NO_PC DO
       E(I) = EP;
       F(I) = FP;
   END # for

   P_Liq = P_Flash;
   P_Vap = P_Flash;

   FOR I:=1 TO NO_COMP DO
       AD(I) = X_Liq(I,1);
       BD(I) = X_Vap(I,1);
   END # for


 END    # Stochastic Flash
 #.................................................

 MODEL Stochastic_Cooler

 PARAMETER
   NO_COMP          AS INTEGER
   NO_PC            AS INTEGER
   AP               AS REAL
   BP               AS REAL
   CP               AS REAL

 VARIABLE
   Flow_Hot_In      AS ARRAY(NO_PC)    OF Molar_Flow
   Flow_Hot_Out     AS ARRAY(NO_PC)    OF Molar_Flow
   P_Hot_In         AS                    Pressure
   P_Hot_Out        AS                    Pressure
   T_Hot_In         AS ARRAY(NO_PC)    OF Temperature
   T_Hot_Out        AS ARRAY(NO_PC)    OF Temperature
   X_Hot_In         AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
   X_Hot_Out        AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
```

```
    T_Cold_In          AS ARRAY(NO_PC)     OF Temperature
    T_Cold_Out         AS ARRAY(NO_PC)     OF Temperature
    Flow_Cold          AS ARRAY(NO_PC)     OF Molar_Flow
    CP_Hot             AS                     Notype0
    CP_Cold            AS                     Notype0
    Latent_Heat        AS                     Notype0
    A                  AS ARRAY(NO_COMP,NO_PC) OF Notype
    B                  AS ARRAY(NO_PC)         OF Notype
    C                  AS ARRAY(NO_PC)         OF Notype
    AD                 AS ARRAY(NO_COMP) OF M_Fraction

  STREAM
    Hot_In      : Flow_Hot_In ,T_Hot_In ,P_Hot_In ,X_Hot_In  AS MainStream
    Hot_Out     : Flow_Hot_Out,T_Hot_Out,P_Hot_Out,X_Hot_Out AS MainStream

  SET
    AP  := 0.0;
    BP  := 0.0;
    CP  := 0.0;

  EQUATION

    FOR I:=1 TO NO_COMP DO

%}

Start coolermodel1

Set [RP] X_Hot_In[I] -> pcall = {{X_Hot_In[I][1,...,9]}};

Set [RV] X_Hot_Out[I] -> pcall = {{X_Hot_Out[I][1,...,9]}};

Models eq1 := X_Hot_Out[I] - X_Hot_In[I]
            = {{A[I][1,...,9]} -> {pcall[1,...,9]}};

End

%{

   END

   FOR I:=1 TO NO_COMP DO
   FOR J:=1 TO NO_PC DO
       A(I,J) = AP;
   END # for j
   END # for i

%}

Start coolermodel2

Set [RP] Flow_Hot_In -> pcall = {{Flow_Hot_In[1,...,9]}},
         T_Hot_In -> pcall = {{T_Hot_In[1,...,9]}},
         T_Cold_In -> pcall = {{T_Cold_In[1,...,9]}},
         Flow_Cold -> pcall = {{Flow_Cold[1,...,9]}},
         T_Cold_Out -> pcall = {{T_Cold_Out[1,...,9]}};

Set [RV] Flow_Hot_Out -> pcall = {{Flow_Hot_Out[1,...,9]}},
         T_Hot_Out -> pcall = {{T_Hot_Out[1,...,9]}};
```

```
Models eq1 := Flow_Hot_Out - Flow_Hot_In
           = {{B[1,...,9]} -> {pcall[1,...,9]}},
      eq2 := Flow_Hot_In*CP_Hot*T_Hot_In - Flow_Hot_In*CP_Hot*T_Hot_Out +
             Flow_Hot_In*Latent_Heat - Flow_Cold*CP_Cold*T_Cold_Out +
             Flow_Cold*CP_Cold*T_Cold_In
           = {{C[1,...,9]} -> {pcall[1,...,9]}};

End

%{

   FOR I:=1 TO NO_PC DO
       B(I) = BP;
       C(I) = CP;
   END # for i

   P_Hot_Out    = P_Hot_In;

   FOR I:=1 TO NO_COMP DO
       AD(I) = X_Hot_Out(I,1);
   END # for

END    # Stochastic Cooler
#................................................

MODEL Stochastic_Heater

PARAMETER
  NO_COMP           AS INTEGER
  NO_PC             AS INTEGER
  AP                AS REAL
  BP                AS REAL
  CP                AS REAL

VARIABLE
  Flow_Cold_In       AS ARRAY(NO_PC)    OF Molar_Flow
  Flow_Cold_Out      AS ARRAY(NO_PC)    OF Molar_Flow
  P_Cold_In          AS                    Pressure
  P_Cold_Out         AS                    Pressure
  T_Cold_In          AS ARRAY(NO_PC)    OF Temperature
  T_Cold_Out         AS ARRAY(NO_PC)    OF Temperature
  X_C_In             AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
  X_C_Out            AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
  CP_Cold            AS                      Notype0
  Latent_Heat        AS                      Notype0
  Heat_Load          AS                      Notype0
  A                  AS ARRAY(NO_COMP,NO_PC) OF Notype
  B                  AS ARRAY(NO_PC)         OF Notype
  C                  AS ARRAY(NO_PC)         OF Notype
  AD                 AS ARRAY(NO_COMP) OF M_Fraction

STREAM
  Cold_In     : Flow_Cold_In ,T_Cold_In ,P_Cold_In ,X_C_In  AS MainStream
  Cold_Out    : Flow_Cold_Out,T_Cold_Out,P_Cold_Out,X_C_Out AS MainStream

SET
  AP  := 0.0;
  BP  := 0.0;
```

```
    CP  := 0.0;

  EQUATION

    FOR I:=1 TO NO_COMP DO

 %}

 Start heatermodel1

 Set [RP] X_C_In[I] -> pcall = {{X_C_In[I][1,...,9]}};

 Set [RV] X_C_Out[I] -> pcall = {{X_C_Out[I][1,...,9]}};

 Models eq1 := X_C_Out[I] - X_C_In[I]
            = {{A[I][1,...,9]} -> {pcall[1,...,9]}};

 End

 %{

    END

    FOR I:=1 TO NO_COMP DO
    FOR J:=1 TO NO_PC DO
        A(I,J) = AP;
    END # for j
    END # for i

 %}

 Start heatermodel2

 Set [RP] Flow_Cold_In -> pcall = {{Flow_Cold_In[1,...,9]}},
          T_Cold_In -> pcall = {{T_Cold_In[1,...,9]}};


 Set [RV] Flow_Cold_Out -> pcall = {{Flow_Cold_Out[1,...,9]}},
          T_Cold_Out -> pcall = {{T_Cold_Out[1,...,9]}};

 Models eq1 := Flow_Cold_Out - Flow_Cold_In
             = {{B[1,...,9]} -> {pcall[1,...,9]}},
        eq2 := Flow_Cold_In*CP_Cold*T_Cold_Out - Flow_Cold_In*CP_Cold*T_Cold_In +
               Flow_Cold_In*Latent_Heat - Heat_Load
             = {{C[1,...,9]} -> {pcall[1,...,9]}};

 End

 %{

    FOR I:=1 TO NO_PC DO
        B(I) = BP;
        C(I) = CP;
    END # for i

    P_Cold_Out    = P_Cold_In;

    FOR I:=1 TO NO_COMP DO
        AD(I) = X_C_Out(I,1);
```

```
  END # for

END    # Stochastic Heater
#.................................................

MODEL Stochastic_Compressor

PARAMETER
  NO_COMP       AS INTEGER
  NO_PC         AS INTEGER
  AP            AS REAL
  BP            AS REAL
  CP            AS REAL

VARIABLE
  Flow_In       AS ARRAY(NO_PC)   OF Molar_Flow
  Flow_Out      AS ARRAY(NO_PC)   OF Molar_Flow
  P_In          AS                   Pressure
  P_Out         AS                   Pressure
  T_In          AS ARRAY(NO_PC)   OF Temperature
  T_Out         AS ARRAY(NO_PC)   OF Temperature
  X_In          AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
  X_Out         AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
  A             AS ARRAY(NO_COMP,NO_PC) OF Notype
  B             AS ARRAY(NO_PC)         OF Notype
  C             AS ARRAY(NO_PC)         OF Notype
  AD            AS ARRAY(NO_COMP) OF M_Fraction

STREAM
  Inlet  : Flow_In ,T_In ,P_In ,X_In  AS  MainStream
  Outlet : Flow_Out,T_Out,P_Out,X_Out AS  MainStream

SET
  AP  := 0.0;
  BP  := 0.0;
  CP  := 0.0;

EQUATION

  FOR I:=1 TO NO_COMP DO

%}

Start compressormodel1

Set [RP] X_In[I] -> pcall = {{X_In[I][1,...,9]}};

Set [RV] X_Out[I] -> pcall = {{X_Out[I][1,...,9]}};

Models eq1 := X_Out[I] - X_In[I]
           = {{A[I][1,...,9]} -> {pcall[1,...,9]}};

End

%{

  END

  FOR I:=1 TO NO_COMP DO
```

```
  FOR J:=1 TO NO_PC DO
      A(I,J) = AP;
  END # for j
  END # for i

%}

Start compressormodel2

Set [RP] Flow_In -> pcall = {{Flow_In[1,...,9]}},
        T_In -> pcall = {{T_In[1,...,9]}};


Set [RV] Flow_Out -> pcall = {{Flow_Out[1,...,9]}},
        T_Out -> pcall = {{T_Out[1,...,9]}};

Models eq1 := Flow_Out - Flow_In
           = {{B[1,...,9]} -> {pcall[1,...,9]}},
       eq2 := T_Out - T_In * SQRT(P_Out/P_In)
           = {{C[1,...,9]} -> {pcall[1,...,9]}};

End

%{

   FOR I:=1 TO NO_PC DO
       B(I) = BP;
       C(I) = CP;
   END # for i

   FOR I:=1 TO NO_COMP DO
       AD(I) = X_Out(I,1);
   END # for

 END    # Stochastic Compressor
 #.....................................................

 MODEL Stochastic_Reactor

 PARAMETER
   NO_COMP      AS INTEGER
   STOICH_COEFF AS ARRAY(NO_COMP) OF REAL
   NO_PC        AS INTEGER
   AP           AS REAL
   BP           AS REAL
   CP           AS REAL
   DP           AS REAL
   EP           AS REAL

 VARIABLE
   Flow_In       AS ARRAY(NO_PC)   OF Molar_Flow
   Flow_Out      AS ARRAY(NO_PC)   OF Molar_Flow
   P_In          AS                   Pressure
   P_Out         AS                   Pressure
   T_In          AS ARRAY(NO_PC)   OF Temperature
   T_Out         AS ARRAY(NO_PC)   OF Temperature
   X_In          AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
   X_Out         AS ARRAY(NO_COMP,NO_PC) OF Mole_Fraction
   BaseConv      AS                 Fraction
```

```
   Conversion    AS ARRAY(NO_PC) OF Fraction
   U_max         AS ARRAY(NO_PC) OF Fraction
   A             AS ARRAY(NO_PC) OF Notype
   B             AS ARRAY(NO_PC) OF Notype
   C             AS ARRAY(NO_PC) OF Notype
   D             AS ARRAY(NO_PC) OF Notype
   E             AS ARRAY(NO_COMP,NO_PC) OF Notype
   AD            AS ARRAY(NO_COMP) OF M_Fraction

 STREAM
   Inlet  : Flow_In ,T_In ,P_In ,X_In  AS  MainStream
   Outlet : Flow_Out,T_Out,P_Out,X_Out AS  MainStream

 SET
   STOICH_COEFF := [-1,-1,2] ;
   AP  := 0.0;
   BP  := 0.0;
   CP  := 0.0;
   DP  := 0.0;
   EP  := 0.0;

 EQUATION

%}

Start reactormodel1

Set [RP] X_In[1] -> pcall = {{X_In[1][1,...,9]}};

Set [RV] U_max -> pcall = {{U_max[1,...,9]}};

Models eq1 := U_max - X_In[1]
            = {{A[1,...,9]} -> {pcall[1,...,9]}};

End

%{

   FOR I:=1 TO NO_PC DO
       A(I) = AP;
   END # for i

%}

Start reactormodel2

Set [RP] Flow_In -> pcall = {{Flow_In[1,...,9]}},
         T_In -> pcall = {{T_In[1,...,9]}};

Set [RV] Flow_Out -> pcall = {{Flow_Out[1,...,9]}},
         T_Out -> pcall = {{T_Out[1,...,9]}},
         Conversion -> pcall = {{Conversion[1,...,9]}};

Models eq1 := Flow_Out - Flow_In
            = {{B[1,...,9]} -> {pcall[1,...,9]}},
       eq2 := T_Out - T_In
            = {{C[1,...,9]} -> {pcall[1,...,9]}},
       eq3 := Conversion - BaseConv - (0.15/600.0)*T_In - (0.1/25.0)*P_In
            = {{D[1,...,9]} -> {pcall[1,...,9]}};
```

```
End

%{

   FOR I:=1 TO NO_PC DO
       B(I) = BP;
       C(I) = CP;
       D(I) = DP;
   END # for i

   FOR I:=1 TO NO_COMP DO

%}

Start reactormodel3

Set [RP] Flow_In -> pcall = {{Flow_In[1,...,9]}},
         X_In[I] -> pcall = {{X_In[I][1,...,9]}},
         Flow_Out -> pcall = {{Flow_Out[1,...,9]}},
         U_max -> pcall = {{U_max[1,...,9]}},
         Conversion -> pcall = {{Conversion[1,...,9]}};

Set [RV] X_Out[I] -> pcall = {{X_Out[I][1,...,9]}};

Models eq1 := Flow_Out*X_Out[I] - Flow_In*X_In[I] -
              Flow_In*U_max*Conversion*STOICH_COEFF[I]
            = {{E[I][1,...,9]} -> {pcall[1,...,9]}};

End

%{

   END # for

   FOR I:=1 TO NO_COMP DO
   FOR J:=1 TO NO_PC DO
       E(I,J) = AP;
   END # for j
   END # for i

   P_Out = P_In;

   FOR I:=1 TO NO_COMP DO
       AD(I) = X_Out(I,1);
   END # for

 END    # Stochastic Reactor
 #.................................................

MODEL Flowsheet

PARAMETER
  NO_COMP  AS INTEGER
  NO_PC    AS INTEGER
  CP_LIQ   AS REAL
  CP_GAS   AS REAL
  T_BUBBLE AS REAL
```

```
UNIT
  C1   AS Stochastic_Compressor        # Makeup Compressor
  C2   AS Stochastic_Compressor        # Recycle Compressor
  S1   AS Stochastic_Splitter          # Feed Splitter
  S2   AS Stochastic_Splitter          # Purge Splitter
  R1   AS Stochastic_Reactor           # Reactor 1
  R2   AS Stochastic_Reactor           # Reactor 2
  M1   AS Stochastic_Mixer             # Recycle/Makeup Mixer
  M2   AS Stochastic_Mixer             # Reaction Product Mixer
  F    AS Stochastic_Flash             # Product Flash
  HX1  AS Stochastic_Cooler            # R1 Product Cooler/Condenser
  HX2  AS Stochastic_Cooler            # R2 Product Cooler/Condenser
  HX3  AS Stochastic_Heater            # Purge Gas Heater
  HX4  AS Stochastic_Heater            # Liquid Product Evaporator

SET
  CP_LIQ        := 0.04   ;
  CP_GAS        := 0.02   ;
  T_BUBBLE      := 330.0  ;
  NO_COMP       :=    3   ;
  NO_PC         :=    9   ;
  M1.NO_STREAMS :=    2   ;
  M2.NO_STREAMS :=    2   ;
  S1.NO_STREAMS :=    2   ;
  S2.NO_STREAMS :=    2   ;

 EQUATION
   M1.Inlets(1)     IS  C1.Outlet;
   M1.Inlets(2)     IS  C2.Outlet;
   M1.Outlet        IS  S1.Inlet;
   S1.Outlets(1)    IS  R1.Inlet;
   S1.Outlets(2)    IS  R2.Inlet;
   R1.Outlet        IS  HX1.Hot_In;
   R2.Outlet        IS  HX2.Hot_In;
   HX1.Hot_Out      IS  M2.Inlets(1);
   HX2.Hot_Out      IS  M2.Inlets(2);
   M2.Outlet        IS  F.Inlet;
   F.Vapour         IS  S2.Inlet;
   S2.Outlets(1)    IS  HX3.Cold_In;
   S2.Outlets(2)    IS  C2.Inlet;
   F.Liquid         IS  HX4.Cold_In;

END # Flowsheet
#.....................................................

MODEL Complete_Plant

UNIT
   Plant AS Flowsheet

EQUATION

  SIGMA( Plant.C1.X_In(,1) )     = 1.    ;

END # Complete_Plant

#=====================================================
```

```
PROCESS example

UNIT
  Plant_total AS Complete_Plant

INPUT

 WITHIN Plant_total DO

 WITHIN Plant DO

    WITHIN HX1 DO
      Flow_Cold(1)              := 400.9        ;
      CP_Cold           := 0.05         ;
      CP_Hot            := CP_GAS       ;
      T_Cold_In(1)              := 298.0        ;
      T_Cold_In(2)              := 1.667        ;
      Latent_Heat       := 0.7          ;
      T_Hot_Out(1)              := T_BUBBLE     ;
          Flow_Cold(2)          := 0.0;
          T_Hot_Out(2)          := 0.0;
      FOR I:=3 TO NO_PC DO
          Flow_Cold(I)          := 0.0;
          T_Cold_In(I)          := 0.0;
          T_Hot_Out(I)          := 0.0;
      END # for
    END  #Within HX1

    WITHIN HX2 DO
      Flow_Cold(1)              := 572.8        ;
      CP_Cold           := 0.05         ;
      CP_Hot            := CP_GAS       ;
      T_Cold_In(1)              := 298.0        ;
      T_Cold_In(2)              := 0.0          ;
      T_Cold_In(3)              := 1.667        ;
      Latent_Heat       := 0.7          ;
      T_Hot_Out(1)              := T_BUBBLE     ;
      FOR I:=2 TO 3 DO
          Flow_Cold(I)          := 0.0;
          T_Hot_Out(I)          := 0.0;
      END # for
      FOR I:=4 TO NO_PC DO
          Flow_Cold(I)          := 0.0;
          T_Cold_In(I)          := 0.0;
          T_Hot_Out(I)          := 0.0;
      END # for
    END  #Within HX2

    WITHIN HX3 DO
      Latent_Heat       := 0.0          ;
      Heat_Load         := 2.4          ;
      CP_Cold           := CP_GAS       ;
    END #Within HX3

    WITHIN HX4 DO
      Latent_Heat       := 0.7          ;
      Heat_Load         := 465.3        ;
      CP_Cold           := CP_LIQ       ;
    END #Within HX4
```

```
    WITHIN F DO
      P_Flash          := 10.0         ;
      T_Flash(1)                := 280.0        ;
      FOR I:=2 TO NO_PC DO
          T_Flash(I)            := 0.0;
      END # for
      K(1)             := 16.0         ;
      K(2)             :=  8.0         ;
      K(3)             :=  0.4         ;
    END #Within F

    WITHIN R1 DO
      BaseConv         := 0.5          ;
    END #Within R1

    WITHIN R2 DO
      BaseConv         := 0.6          ;
    END #Within R2

    WITHIN C1 DO
      X_In(1,1)        := 0.475        ;
      X_In(1,2)        := 0.0          ;
      X_In(1,3)        := 0.0          ;
      X_In(1,4)        := 0.00833      ;
      X_In(1,5)        := 0.0          ;
      X_In(3,1)        := 0.0          ;
      Flow_In(1)               := 100.0        ;
      Flow_In(2)               := 0.0          ;
      Flow_In(3)               := 0.0          ;
      Flow_In(4)               := 0.0          ;
      Flow_In(5)               := 3.33         ;
      T_In(1)          := 300.0        ;
      P_In             := 5.0          ;
      P_Out            := 22.5         ;
      FOR I:=2 TO 5 DO
          X_In(2,I)            := 0.0;
          X_In(3,I)            := 0.0;
          T_In(I)              := 0.0;
      END # for
      FOR I:=6 TO NO_PC DO
          X_In(1,I)            := 0.0;
          X_In(2,I)            := 0.0;
          X_In(3,I)            := 0.0;
          Flow_In(I)           := 0.0;
          T_In(I)              := 0.0;
      END # for
    END #Within C1

    WITHIN C2 DO
      P_Out            := 22.5         ;
    END #Within C1

    S1.Fraction_To_1 := 0.412;
    S2.Fraction_To_1 := 0.05;

END #Within Plant

END #Within Plant_total
```

```
INITIAL
   STEADY_STATE

SCHEDULE
 SEQUENCE

  RESET

        WITHIN Plant_total DO

 #-------------------------------------------------------------------
          Plant.C1.X_In(1,1)       := 0.475 ;
 #-------------------------------------------------------------------

         END # within

  END # reset

 END # Sequence

END # process example
#.....................................................................

%}

%%
```

F.4 Multiple stationary points example

F.4.1 file: massactions.dem

The following code is an input file written in the prototype language of DEMM. This input file describes the stochastic model of mass actions kinetics that is represented by a set of differential equations.

```
# example for mass action kinetics model
# see Willamowski and Rossler in Z. Naturforsch. 35 a, 1980,
# pp. 317-318.

%{
      external fex
      double precision atol, rtol, rwork, t, tout, y, jdum
      dimension y(24), rwork(814), iwork(44)
      neq = 24

c     Define initial conditions

      y(1) = 12.d0
      y(2) = 0.d0
      y(3) = 0.d0
      y(4) = 0.d0
      y(5) = 0.d0
      y(6) = 0.d0
      y(7) = 0.d0
      y(8) = 0.d0
      y(9) = 22.d0
```

```
      y(10) = 0.d0
      y(11) = 2.d0
      y(12) = 0.d0
      y(13) = 0.d0
      y(14) = 0.d0
      y(15) = 0.d0
      y(16) = 0.d0
      y(17) = 8.d0
      y(18) = 0.d0
      y(19) = 0.d0
      y(20) = 0.d0
      y(21) = 0.d0
      y(22) = 0.d0
      y(23) = 0.d0
      y(24) = 0.d0

c     Define time steps and flag variables

      t = 0.d0
      tout = .1d0
      itol = 1
      rtol = 1.d-8
      atol = 1.d-12
      itask = 1
      istate = 1
      iopt = 0
      lrw = 814
      liw = 44
      mf = 22
      iwrit = 1
      do 40 iout = 1,100

c     Use LSODE to calculate the dependent variables

        call lsode(fex,neq,y,t,tout,itol,rtol,atol,itask,istate,
     1     iopt,rwork,lrw,iwork,liw,jdum,mf)
        if (iwrit .gt. 0) go to 25

c     Print the expected value for x, y, and z

          write(6,20)t,y(1),y(9),y(17)
   20   format(7h at t =,e12.4,6h   y =,3e14.6)
   25   iwrit = iwrit + 1
        if (iwrit .eq. 10) iwrit = 0
        if (istate .lt. 0) go to 80
        t = tout
   40   tout = tout+0.1d0
      stop
   80 write(6,90)istate
   90 format(///22h error halt.. istate =,i3)
      stop
      end

c     Define the model

      subroutine fex (neq, t, y, ydot)
      double precision t, y, ydot
      dimension y(24), ydot(24)
      double precision a1,a2,a3,a4,a5
```

```
      double precision k1,k20,k21,k22,k23,k5

c     Define the values of parameters

      a1 = 30.0
      a2 = 0.01
      a3 = 0.01
      a4 = 16.5
      a5 = 10.0
      k1 = 0.25
      k20 = 0.058
      k21 = 0.005
      k22 = 0.0008
      k23 = 0.0001
      k5 = 0.5

%}

# define the format of output file

%Format fortran

# begin the stochastic modeling

%%

# define the coordinates for this problem
# "e1" and "e2" are independent standard Gaussian random variables

Set [BF] basis = {e1,e2};

# declare the values of moments of "e1" and "e2"
# in this case, we give the first fourteen

Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135};

# declare the form of polynomial chaos expansion
# since we use Gaussian random variable as the coordinate
# then the polynomial chaos expansion uses Hermite polynomials.
# in this case, we use up to the first four terms of
# polynomial chaos expansion for the uncertain parameter
# and eight terms for dependent variables

Set [PC] pcp = {1,e1,e1^2-1,e1^3-3*e1},
         pcr = {1,e1,e2,0.707107*(e1^2-1),e1*e2,0.707107*(e2^2-1),
                0.408248*(e1^3-3*e1),0.408248*(e2^3-3*e2)};

# define k2 as uncertain parameter, and assume that
# we know its four coefficients of polynomial chaos expansion

Set [RP] k2 -> pcp = {k20,k21,k22,k23};

# x,y, and z are dependent variables, and now become
# uncertain variables

Set [RV] x -> pcr = {{y[1,...,8]}},
         y -> pcr = {{y[9,...,16]}},
         z -> pcr = {{y[17,...,24]}};
```

```
# define the name of file for storing the deterministic
# equivalent model

Start massmodel

# mass action kinetics equations for x, y, and z

Models eq1 := x*(a1-k1*x-z-y)+k2*y^2+a3 =
              {{ydot[1,...,8]} -> {pcr[1,...,8]}},
       eq2 := y*(x-k2*y-a5)+a2 =
              {{ydot[9,...,16]} -> {pcr[1,...,8]}},
       eq3 := z*(a4-x-k5*z)+a3 =
              {{ydot[17,...,24]} -> {pcr[1,...,8]}};

End

# end of the stochastic modeling

%%
      return
      end
```

F.5 Generalized mechanism for photochemical smog

F.5.1 file: mechanisms.dem

The following code is an input file written in the prototype language of DEMM. This input file describes the stochastic model of photochemical smog that is represented by a set of differential equations.

```
# example for a generalized reaction mechanism for
# photochemical smog model
# taken from: J.H. Seinfeld, "Atmospheric Chemistry and
# Physics of Air Pollution", pages 158 - 162.

%{

      external fex
      double precision atol, rtol, rwork, t, tout, y, jdum
      double precision std1,std2,std3,std4,std5
      dimension y(300), rwork(92722), iwork(320)
      neq = 300

c     Define initial conditions

      do 10 i = 1,300
   10    y(i) = 0.0d0

      y(1) = 2.0d0
      y(26) = 2.0d0
      y(51) = 0.5d0
      y(76) = 0.1d0
      y(101) = 0.d0
      y(276) = 209460.0d0

c     Define time steps and flag variables
```

```
      t = 0.d0
      tout = .1d0
      itol = 1
      rtol = 1.d-14
      atol = 1.d-14
      itask = 1
      istate = 1
      iopt = 0
      lrw = 92722
      liw = 320
      mf = 22
      iwrit = 1

      std1 = 0.0d0
      std2 = 0.0d0
      std3 = 0.0d0
      std4 = 0.0d0
      std5 = 0.0d0
      write(6,20)t,y(1),std1,y(26),std2,y(51),std3,
     +           y(76),std4,y(101),std5

      do 40 iout = 1,6000

c     Use LSODE to calculate the dependent variables

        call lsode(fex,neq,y,t,tout,itol,rtol,atol,itask,istate,
     1     iopt,rwork,lrw,iwork,liw,jdum,mf)
        if (iwrit .gt. 0) go to 25

c     Print the expected values and standard deviations

      std1 = 0.0d0
      std2 = 0.0d0
      std3 = 0.0d0
      std4 = 0.0d0
      std5 = 0.0d0
      do 15 i = 2,25
         std1 = std1 + y(i)**2.0
         std2 = std2 + y(25+i)**2.0
         std3 = std3 + y(50+i)**2.0
         std4 = std4 + y(75+i)**2.0
 15      std5 = std5 + y(100+i)**2.0
      std1 = sqrt(std1)
      std2 = sqrt(std2)
      std3 = sqrt(std3)
      std4 = sqrt(std4)
      std5 = sqrt(std5)

           write(6,20)t,y(1),std1,y(26),std2,y(51),std3,
     +                y(76),std4,y(101),std5

 20     format(e12.4,10e14.6)
 25     iwrit = iwrit + 1
        if (iwrit .eq. 100) iwrit = 0
        if (istate .lt. 0) go to 80
        t = tout
 40     tout = tout+0.1d0
      stop
```

```
  80  write(6,90)istate
  90  format(///22h error halt.. istate =,i3)
      stop
      end

c     Define the model

      subroutine fex (neq, t, y, ydot)
      double precision t, y, ydot
      dimension y(300), ydot(300)
      double precision k10,k20,k30,k40,k50,k60,k70,k80
      double precision k90,k100,k110,k120
      double precision k11,k21,k31,k41,k51,k61,k71,k81
      double precision k91,k101,k111,k121

c     Define the values of parameters +/- 20 % std

      k10 = 0.533
      k11 = 0.1066
      k20 = 2.183d-5
      k21 = 4.366d-6
      k30 = 26.59
      k31 = 5.318
      k40 = 3.775d3
      k41 = 755.0d0
      k50 = 2.341d4
      k51 = 4682.0d0
      k60 = 1.91d-4
      k61 = 0.0000382d0
      k70 = 1.214d4
      k71 = 2428.0d0
      k80 = 1.127d4
      k81 = 2254.0d0
      k90 = 1.127d4
      k91 = 2254.0d0
      k100 = 1.613d4
      k101 = 3226.0d0
      k110 = 6.893d3
      k111 = 1378.6d0
      k120 = 2.143d-2
      k121 = 0.004286d0

%}

# define the format of output file

%Format fortran

# begin the stochastic modeling

%%

# define the coordinates for this problem
# "e1" - "e12" are independent standard Gaussian random variables

Set [BF] basis = {e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12};

# declare the values of moments of "e1" - "e12"
# in this case, we give the first fourteen
```

```
Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135};

# declare the form of polynomial chaos expansion
# since we use Gaussian random variable as the coordinate
# then the polynomial chaos expansion uses Hermite polynomials.

Set [PC] pcr = {1,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,
                0.707107*(e1^2-1),0.707107*(e2^2-1),
                0.707107*(e3^2-1),0.707107*(e4^2-1),
                0.707107*(e5^2-1),0.707107*(e6^2-1),
                0.707107*(e7^2-1),0.707107*(e8^2-1),
                0.707107*(e9^2-1),0.707107*(e10^2-1),
                0.707107*(e11^2-1),0.707107*(e12^2-1)
                };

# note:
# the constants will influence the value of higher moments
# for example,
# y = y0 + y1 e1 + y2 0.707107*(e1^2-1) + y3 0.408248*(e1^3-3*e1)
# so the mean value is still y0,
# but the variance is not y1^2 + 2 y2^2 + 6 y3^2
# now the variance is y1^2 + y2^2 + y3^2

# define k1 - k12 as uncertain parameters, and assume that
# we know their two coefficients of polynomial chaos expansions

Set [RP] k1 := k10 + k11*e1,
         k2 := k20 + k21*e2,
         k3 := k30 + k31*e3,
         k4 := k40 + k41*e4,
         k5 := k50 + k51*e5,
         k6 := k60 + k61*e6,
         k7 := k70 + k71*e7,
         k8 := k80 + k81*e8,
         k9 := k90 + k91*e9,
         k10n := k100 + k101*e10,
         k11n := k110 + k111*e11,
         k12n := k120 + k121*e12;

# define all uncertain variables

Set [RV] y1 -> pcr = {{y[1,...,25]}},
         y2 -> pcr = {{y[26,...,50]}},
         y3 -> pcr = {{y[51,...,75]}},
         y4 -> pcr = {{y[76,...,100]}},
         y5 -> pcr = {{y[101,...,125]}},
         y6 -> pcr = {{y[126,...,150]}},
         y7 -> pcr = {{y[151,...,175]}},
         y8 -> pcr = {{y[176,...,200]}},
         y9 -> pcr = {{y[201,...,225]}},
         y10 -> pcr = {{y[226,...,250]}},
         y11 -> pcr = {{y[251,...,275]}},
         y12 -> pcr = {{y[276,...,300]}};

# define the names of files for storing the deterministic
# equivalent models
```

```
Start mechanismsmodel1

Models eq1 := - k4*y1*y6 =
             {{ydot[1,...,25]} -> {pcr[1,...,25]}},
       eq2 := - k5*y2*y6 - k6*y2 + k8*y7*y3 =
             {{ydot[26,...,50]} -> {pcr[1,...,25]}},
       eq3 := - k3*y3*y5 - k7*y8*y3
              - k8*y7*y3 - k9*y9*y3
              + k1*y4 =
             {{ydot[51,...,75]} -> {pcr[1,...,25]}};

End

Start mechanismsmodel2

Models eq4 := - k1*y4 - k10n*y6*y4
              - k11n*y9*y4 + k3*y3*y5
              + k7*y8*y3 + k8*y7*y3
              + k9*y9*y3 + k12n*y10 =
             {{ydot[76,...,100]} -> {pcr[1,...,25]}},
       eq5 := - k3*y3*y5 + k2*y11*y12 =
             {{ydot[101,...,125]} -> {pcr[1,...,25]}},
       eq6 := - k4*y1*y6 - k5*y2*y6
              - k10n*y6*y4 + k7*y8*y3 =
             {{ydot[126,...,150]} -> {pcr[1,...,25]}};

End

Start mechanismsmodel3

Models eq7 := - k8*y7*y3 + k4*y1*y6
              + k6*y2 + k9*y9*y3 =
             {{ydot[151,...,175]} -> {pcr[1,...,25]}},
       eq8 := - k7*y8*y3 + k6*y2
              + k8*y7*y3 =
             {{ydot[176,...,200]} -> {pcr[1,...,25]}},
       eq9 := - k9*y9*y3 - k11n*y9*y4
              + k5*y2*y6 + k12n*y10 =
             {{ydot[201,...,225]} -> {pcr[1,...,25]}};

End



Start mechanismsmodel4

Models eq10 := - k12n*y10 + k11n*y9*y4 =
             {{ydot[226,...,250]} -> {pcr[1,...,25]}},
       eq11 := - k2*y11*y12 + k1*y4 =
             {{ydot[251,...,275]} -> {pcr[1,...,25]}},
       eq12 := - k2*y11*y12 + k3*y3*y5 =
             {{ydot[276,...,300]} -> {pcr[1,...,25]}};

End

# end of the stochastic modeling

%%
```

```
      return
      end
```

F.6 Reduced H_2/O_2 mechanism

F.6.1 file: h2o2example.dem

The following code is an input file written in the prototype language of DEMM. This input file describes the stochastic model of H_2/O_2 reduced mechanism that is represented by a set of differential equations.

```
# example for reduced H2/O2 mechanism
# 10 reactions, 7 species

%{

      external fex
      double precision atol, rtol, rwork, t, tout, y, jdum
      double precision std1,std2,std3,std4,std5,std6,std7
      dimension y(147), rwork(22954), iwork(167)
      neq = 147

c     Define initial conditions

      do 10 i = 1,147
   10    y(i) = 0.0d0

      y(1) = 0.0d0
      y(22) = 2.06d-6
      y(43) = 1.04d-6
      y(64) = 0.0d0
      y(85) = 4.285d-3
      y(106) = 0.0d0
      y(127) = 0.0d0

c     Define time steps and flag variables

      t = 0.d0
      tout = .01d0
      itol = 1
      rtol = 1.d-14
      atol = 1.d-14
      itask = 1
      istate = 1
      iopt = 0
      lrw = 22954
      liw = 167
      mf = 22
      iwrit = 1

      std1 = 0.0d0
      std2 = 0.0d0
      std3 = 0.0d0
      std4 = 0.0d0
      std5 = 0.0d0
      std6 = 0.0d0
      std7 = 0.0d0
```

```
      write(6,20)t,y(1),std1,y(22),std2,y(43),std3,
     +           y(64),std4,y(85),std5,y(106),std6,
     +           y(127),std7

      do 40 iout = 1,500

c     Use LSODE to calculate the dependent variables

        call lsode(fex,neq,y,t,tout,itol,rtol,atol,itask,istate,
     1     iopt,rwork,lrw,iwork,liw,jdum,mf)
        if (iwrit .gt. 0) go to 25

c     Print the expected values and standard deviations

      std1 = 0.0d0
      std2 = 0.0d0
      std3 = 0.0d0
      std4 = 0.0d0
      std5 = 0.0d0
      std6 = 0.0d0
      std7 = 0.0d0
      do 15 i = 2,21
         std1 = std1 + y(i)**2.0
         std2 = std2 + y(21+i)**2.0
         std3 = std3 + y(42+i)**2.0
         std4 = std4 + y(63+i)**2.0
         std5 = std5 + y(84+i)**2.0
         std6 = std6 + y(105+i)**2.0
 15      std7 = std7 + y(126+i)**2.0
      std1 = sqrt(std1)
      std2 = sqrt(std2)
      std3 = sqrt(std3)
      std4 = sqrt(std4)
      std5 = sqrt(std5)
      std6 = sqrt(std6)
      std7 = sqrt(std7)
      write(6,20)t,y(1),std1,y(22),std2,y(43),std3,
     +           y(64),std4,y(85),std5,y(106),std6,
     +           y(127),std7

 20     format(e12.4,14e14.6)
 25     iwrit = iwrit + 1
        if (iwrit .eq. 100) iwrit = 0
        if (istate .lt. 0) go to 80
        t = tout
 40     tout = tout+0.01d0
      stop
 80   write(6,90)istate
 90   format(///22h error halt.. istate =,i3)
      stop
      end

c     Define the model

      subroutine fex (neq, t, y, ydot)
      double precision t, y, ydot
      dimension y(147), ydot(147)
      double precision k10,k20,k30,k40,k50,k60,k70,k80
      double precision k90,k100
```

```
      double precision k11,k21,k31,k41,k51,k61,k71,k81
      double precision k91,k101

c     Define the values of parameters

      k10 = 6.7628d11
      k11 = 4.25718d10
      k20 = 2.2426d-12
      k21 = 3.54331d-13
      k30 = 1.0723d13
      k31 = 1.69423d12
      k40 = 2.6608d13
      k41 = 4.20406d12
      k50 = 4.8641d-5
      k51 = 0.0d0
      k60 = 2.0196d12
      k61 = 1.59548d11
      k70 = 1.2294d4
      k71 = 0.0d0
      k80 = 6.7957d11
      k81 = 4.79097d10
      k90 = 4.0515d1
      k91 = 6.40137d0
      k100 = 7.3644d6
      k101 = 7.3644d5

%}

# define the format of output file

%Format fortran

# begin the stochastic modeling

%%

# define the coordinates for this problem
# "e1" - "e10" are independent standard Gaussian random variables

Set [BF] basis = {e1,e2,e3,e4,e5,e6,e7,e8,e9,e10};

# declare the values of moments of "e1" - "e10"
# in this case, we give the first fourteen

Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135};

# declare the form of polynomial chaos expansion
# since we use Gaussian random variable as the coordinate
# then the polynomial chaos expansion uses Hermite polynomials.

Set [PC] pcr = {1,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,
                0.707107*(e1^2-1),0.707107*(e2^2-1),
                0.707107*(e3^2-1),0.707107*(e4^2-1),
                0.707107*(e5^2-1),0.707107*(e6^2-1),
                0.707107*(e7^2-1),0.707107*(e8^2-1),
                0.707107*(e9^2-1),0.707107*(e10^2-1)};

# note:
```

```
# the constants will influence the value of higher moments
# for example,
# y = y0 + y1 e1 + y2 0.707107*(e1^2-1) + y3 0.408248*(e1^3-3*e1)
# so the mean value is still y0,
# but the variance is not y1^2 + 2 y2^2 + 6 y3^2
# now the variance is y1^2 + y2^2 + y3^2

# define k1 - k10 as uncertain parameters, and assume that
# we know their two coefficients of polynomial chaos expansions

Set [RP] k1 := k10 + k11*e1,
         k2 := k20 + k21*e2,
         k3 := k30 + k31*e3,
         k4 := k40 + k41*e4,
         k5 := k50 + k51*e5,
         k6 := k60 + k61*e6,
         k7 := k70 + k71*e7,
         k8 := k80 + k81*e8,
         k9 := k90 + k91*e9,
         k10n := k100 + k101*e10;

# define all uncertain variables

Set [RV] y1 -> pcr = {{y[1,...,21]}},
         y2 -> pcr = {{y[22,...,42]}},
         y3 -> pcr = {{y[43,...,63]}},
         y4 -> pcr = {{y[64,...,84]}},
         y5 -> pcr = {{y[85,...,105]}},
         y6 -> pcr = {{y[106,...,126]}},
         y7 -> pcr = {{y[127,...,147]}};

# define the name of file for storing the deterministic
# equivalent model



# simple nonlinear kinetics equations for x, y and z

Start h2o2model1

Models eq1 := k1*y2*y4 + k2*y5 - k3*y1*y3 + k10n*y2*y6 =
              {{ydot[1,...,21]} -> {pcr[1,...,21]}},
       eq2 := -k1*y2*y4 - k10n*y2*y6 =
              {{ydot[22,...,42]} -> {pcr[1,...,21]}},
       eq3 := -k3*y1*y3 + k4*y4*y6 - k5*y5*y3 + k8*y6*y6 =
              {{ydot[43,...,63]} -> {pcr[1,...,21]}};

End

Start h2o2model2

Models eq4 := -k1*y2*y4 + k2*y5 - k4*y4*y6 +
               k5*y5*y3 - k6*y7*y4 + k7*y5*y6 +
               2.0*k9*y7 =
              {{ydot[64,...,84]} -> {pcr[1,...,21]}},
       eq5 := k1*y2*y4 - k2*y5 + k4*y4*y6 - k5*y5*y3 +
               k6*y7*y4 - k7*y5*y6 =
              {{ydot[85,...,105]} -> {pcr[1,...,21]}},
       eq6 := k3*y1*y3 - k4*y6*y4 + k5*y5*y3 +
```

```
            k6*y7*y4 - k7*y5*y6 - 2.0*k8*y6*y6 - k10n*y2*y6 =
           {{ydot[106,...,126]} -> {pcr[1,...,21]}};

End

Start h2o2model3

Models eq7 := -k6*y7*y4 + k7*y5*y6 + k8*y6*y6 - k9*y7 +
              k10n*y2*y6 =
             {{ydot[127,...,147]} -> {pcr[1,...,21]}};

End

# end of the stochastic modeling

%%

      return
      end
```

F.7 Reactors sequence problem

F.7.1 file: sreactor.dem

The following code is an input file written in the prototype language of DEMM. This input file describes the stochastic model of a series of reactors that is represented by an optimization model in the GAMS environment.

```
# stochastic optimization example
# two reactor in series problem
# see Manousiothakis and Sourlas in Chem. Eng. Comm., 115:127-147
# 1992 (note: there are errors in their deterministic model, and
#             there are actually 3 local optima instead of 2 as
#             stated in their paper,
#             see Menner A. Tatang's Ph.D. proposal)

%{
$TITLE Stochastic optimization example

$OFFSYMXREF
$OFFSYMLIST

* Parameter declarations:

PARAMETERS

   KA10        mean value of reaction rate constant of A in
               reactor 1
   KA11        standard deviation of reaction rate constant of A
               in reactor 1
   KA2         reaction rate constant of A in reactor 2
   KB1         reaction rate constant of B in reactor 1
   KB2         reaction rate constant of B in reactor 2
   CA0         inlet concentration of A to reactor 1;

* and their values
```

```
KA10 = 0.096540;
KA11 = 0.03;
KA2 = 0.097515;
KB1 = 0.035272;
KB2 = 0.039191;
CA0 = 1.0;

* Variable declarations:

VARIABLES

   OBV          objective function value
   CA11         second term of concentration of A from reactor 1
   CA12         third term of concentration of A from reactor 1
   CA21         second term of concentration of A from reactor 2
   CA22         third term of concentration of A from reactor 2
   CB11         second term of concentration of B from reactor 1
   CB12         third term of concentration of B from reactor 1
   CB21         second term of concentration of B from reactor 2
   CB22         third term of concentration of B from reactor 2;

POSITIVE VARIABLES

   CA10         first term of concentration of A from reactor 1
   CA20         first term of concentration of A from reactor 2
   CB10         first term of concentration of B from reactor 1
   CB20         first term of concentration of B from reactor 2
   V1           residence time of reactor 1
   V2           residence time of reactor 2
   Z1           dummy variable 1
   Z2           dummy variable 2;

EQUATIONS

   E11          equality constraint 1A
   E12          equality constraint 1B
   E13          equality constraint 1C
   E21          equality constraint 2A
   E22          equality constraint 2B
   E23          equality constraint 2C
   E31          equality constraint 3A
   E32          equality constraint 3B
   E33          equality constraint 3C
   E41          equality constraint 4A
   E42          equality constraint 4B
   E43          equality constraint 4C
   E5           equality constraint 5
   E6           equality constraint 6
   G1           inequality constraint 1
   OB           objective equation;

%}

# define the format of output files

%Format gams

# begin the stochastic modeling
```

```
%%

# set a symbol for Gaussian random variable
# as a coordinate in the probability space,
# in this case "x" is a standard Gaussian random variable

Set [BF] basis = {x};

# declare the values of moments of "x"
# in this case, we give the first fourteen

Set [MV] expected -> basis = {0,1,0,3,0,15,0,105,0,
                              945,0,10395,0,135135};

# declare the form of polynomial chaos expansion
# since we use Gaussian random variable as the coordinate
# then the polynomial chaos expansion uses Hermite polynomials.
# in this case, we use up to the first three terms of
# polynomial chaos expansion

Set [PC] polychaos = {1,x,x^2-1};

# define the reaction rate constant in the first reactor as
# a Gausssian random variable with mean value "KA10" and
# standard deviation "KA11"

Set [RP] KA1 := KA10 + KA11*x;

# define the symbols for coefficients in the polynomial chaos
# expansions for each dependent or state variable

Set [RV] CA1 -> polychaos = {CA10,CA11,CA12},
         CA2 -> polychaos = {CA20,CA21,CA22},
         CB1 -> polychaos = {CB10,CB11,CB12},
         CB2 -> polychaos = {CB20,CB21,CB22};

# define the name of file for storing the deterministic
# equivalent model

Start series

# mass balances and steady state formulations as
# equality constraints

Models eq1 := CA1 - CA0 + KA1*CA1*V1 =
              {{E11,E12,E13} -> {polychaos[1,...,3]}},
       eq2 := CA2 - CA1 + KA2*CA2*V2 =
              {{E21,E22,E23} -> {polychaos[1,...,3]}},
       eq3 := CB1 + CA1 - CA0 + KB1*CB1*V1 =
              {{E31,E32,E33} -> {polychaos[1,...,3]}},
       eq4 := CB2 - CB1 + CA2 - CA1 + KB2*CB2*V2 =
              {{E41,E42,E43} -> {polychaos[1,...,3]}};

End

# end of the stochastic modeling

%{
```

```
* Define the limit of cost horizon, and objective function

E5..  Z1*Z1 - V1 =E= 0;
E6..  Z2*Z2 - V2 =E= 0;
G1..  Z1 + Z2 =L= 4.0;
OB..  OBV =E= CB20;

%}

%%

* Give initial guesses

CA10.L = 0.7;
CA11.L = -0.05;
CA12.L = 0.006;
CA20.L = 0.5;
CA21.L = -0.05;
CA22.L = 0.005;
CB10.L = 0.25;
CB11.L = 0.06;
CB12.L = -0.005;
CB20.L = 0.38;
CB21.L = 0.04;
CB22.L = -0.001;
V1.L = 4.0;
V2.L = 4.0;
Z1.L = 2.0;
Z2.L = 2.0;

OPTION LIMROW = 0;
OPTION LIMCOL = 0;

MODEL REACTORS /ALL/;
SOLVE REACTORS USING NLP MAXIMIZING OBV;
```

Bibliography

[1] G. Adomain. Stochastic system analysis. In G. Adomian, editor, *Applied Stochastic Processes*, pages 1–17, New York, 1980. Academic Press, Inc.

[2] G. Adomian and R. Rach. Generalization of adomian polynomials to functions of several variables. *Computers Math. Applic.*, 24(5/6):11–24, 1992.

[3] N. Agmon, Y. Alhassid, and R.D. Levine. An algorithm for finding the distribution of maximal entropy. *Journal of Computational Physics*, 30:250–258, 1979.

[4] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[5] R.S. Andsten and J.K. Vaurio. Sensitivity, uncertainty, and importance analysis of a risk assessment. *Nuclear Technology*, 98:160–170, 1992.

[6] N. Aubry, R. Guyonnet, and R. Lima. Spatiotemporal analysis of complex signals: Theory and applications. *J. Stat. Phys.*, 64(3/4):683–739, 1991.

[7] N. Aubry, W.-Y. Lian, and E.S. Titi. Preserving symmetries in the proper orthogonal decomposition. *SIAM J. Sci. Comput.*, 14(2):483–505, 1993.

[8] G.A. Baker. Transformation of non-normal frequency distributions into normal distributions. *Annals of Mathematical Statistics*, 5:113–123, 1934.

[9] H. Bandemer. Specification of fuzzy data. In H. Bandemer, editor, *Modelling Uncertain Data*, pages 62–68. Akademie Verlag, Berlin, 1993.

[10] P.I. Barton. *The Modelling and Simulation of Combined Discrete/Continuous Processes.* Ph.D. thesis, Imperial College of Science, Technology and Medicine, 1992.

[11] R.A. Becker, J.M. Chambers, and A.R. Wilks. *The New S Language, A Programming Environment for Data Analysis and Graphics.* Wadsworth & Brooks/Cole, California, 1988.

[12] P. Beckmann. *Orthogonal Polynomials for Engineers and Physicists.* Golem Press., 1973.

[13] B. Bereanu. On stochastic linear programming. the laplace transform of the distribution of the optimum and applications. *Journal of Mathematical Analysis and Applications*, 15:280–294, 1966.

[14] B. Bereanu. Some numerical methods in stochastic linear programming under risk and uncertainty. In M.A.H. Dempster, editor, *Stochastic Programming*, pages 196–205, London, 1980. Academic Press.

[15] R.A. Blau. Stochastic programming and decision analysis: An apparent dilemma. *Management Science*, 21(3):271–276, 1974.

[16] A. Borison and H. Mueller. Forecasting fuel requirements uncertainty. *IEEE Transactions on Power Systems*, 3(3):1046–1051, 1988.

[17] G.E.P. Box and N.R. Draper. The bayesian estimation of common parameters from several responses. *Biometrika*, 52(3 and 4):355–365, 1965.

[18] G.E.P. Box and N.R. Draper. *Empirical Model-Building and Response Surface.* Wiley, New York, 1986.

[19] G.E.P. Box and M.E. Müller. A note on the generation of random normal deviates. *Ann. Math. Statist*, 26:610–611, 1958.

[20] K. Breitung. Probability approximations by log-likelihood maximization. *Journal of Engineering Mechanics*, 117(3):457–477, 1991.

[21] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS A User's Guide.* The Scientific Press, 1988.

[22] S.T. Buckland. Fitting density functions with polynomials. *Applied Statistics*, 41(1):63–76, 1992.

[23] N.P. Buslenko, D.I. Golenko, Y.A. Shreider, I.M. Sobol, and V.G. Sragovich. *The Monte Carlo Method: The Method of Statistical Trials.* Pergamon Press, New York, 1967.

[24] R.H. Cameron and W.T. Martin. The orthogonal development of non-linear functionals in series of fourier-hermite functionals. *Ann. Math.*, 48(2):385–392, 1947.

[25] P. Canu and W.H. Ray. Discrete weighted residual methods applied to polymerization reactions. *Computers and Chem. Engng.*, 15(8):549–564, 1991.

[26] C. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods in Fluid Dynamics.* Springer-Verlag, New York, 1988.

[27] B.D. Carter. *On the Probability Distribution of Rational Functions of Independent H-Function Variates.* Ph.D. thesis, University of Arkansas, 1970.

[28] J.M. Chambers. On methods of asymptotic approximation for multivariate distributions. *Biometrika*, 54(3-4):367–383, 1967.

[29] R.J. Charlson, S.E. Schwartz, J.M. Hales, R.D. Cess, J.A. Coakley Jr., J.E. Hansen, and D.J. Hofmann. Climate forcing by anthropogenic aerosols. *Science*, 255:423–430, 1992.

[30] A. Charnes and W.W. Cooper. Chance-constrained programming. *Management Sci.*, 6(1):73–79, 1959.

[31] A. Charnes and W.W. Cooper. Deterministic equivalents for optimizing and satisficing under chance constraints. *Operations Res.*, 11:18–39, 1963.

[32] S.W. Churchill. The role of analysis in the rate processes. *Industrial & Engineering Chemistry Research*, 31:643–658, 1992.

[33] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the "Simulated Annealing" algorithm. *ACM Trans. Math. Soft.*, 13(3):262–280, 1987.

[34] J. Dagpunar. *Principles of Random Variate Generation.* Oxford University Press, Oxford, 1988.

[35] R.W. Derksen and P.J. Sullivan. Moment approximations for probability density functions. *Combustion and Flame*, 81:378–391, 1990.

[36] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science Vol. B, Formal Models and Semantics*, pages 243–320, Amsterdam, 1990. Elsevier.

[37] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, New York, 1986.

[38] U.M. Diwekar and E.S. Rubin. Stochastic modeling of chemical processes. *Computers Chem. Engng.*, 15(2):105–114, 1991.

[39] J.D. Doll and D.L. Freeman. Randomly exact methods. *Science*, 234:1356–1360, 1986.

[40] W.M. Dong and F.S. Wong. Fuzzy weighted averages and implementation of the extension principle. *Fuzzy Sets and Systems*, 21:183–199, 1987.

[41] C. Donnelly and R. Stallman. *Bison, the YACC-compatible Parser Generator.* Free Software Foundation, Cambridge, Massachusetts, 1991.

[42] F.Y. Edgeworth. On the representation of statistical frequency by a series. *Journal of the Royal Statistical Society, Series A*, 77:102–106, 1907.

[43] F.Y. Edgeworth. On the use of analytical geometry to represent certain kinds of statistics. *Journal of the Royal Statistical Society*, 77:300–312, 1914.

[44] W.P. Elderton and N.L. Johnson. *Systems of Frequency Curves.* Cambridge University Press, London, 1969.

[45] Y. Ermoliev and R.J.B. Wets. *Numerical Techniques for Stochastic Optimization.* Springer-Verlag, New York, 1988.

[46] G. Fichtner, H.J. Reinhart, and D.W.T. Rippin. The design of flexible chemical plants by the applications of interval mathematics. *Computers and Chemical Engineering*, 14(11):1311–1316, 1990.

[47] C.N. Fischer and R.J, Jr. LeBlanc. *Crafting a Compiler.* The Benjamin/Cummings Publishing Co., Menlo Park, California, 1988.

[48] R.J. Fleming. The dynamics of uncertainty: Application to parameterization constants in climate models. *Climate Dynamics*, 8:135–150, 1993.

[49] C. Fox. The G and H-functions as symmetrical Fourier kernels. *Transactions of the American Mathematical Society*, 98:395–429, 1961.

[50] B.S. Garbow, J.M. Boyle, J.J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension.* Springer-Verlag, New York, 1977.

[51] C.W. Gardiner. *Handbook of Stochastic Methods for Physics, Chemistry, and the Natural Sciences.* Springer-Verlag, Berlin, 1985.

[52] W. Gautschi. Algorithm 726: ORTHPOL - a package of routines for generating orthogonal polynomials and gauss-Type quadrature rules. *ACM Transactions on Mathematical Software*, 20(1):21–62, 1994.

[53] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, (6):721–741, 1984.

[54] J. Geweke. Modelling nonlinearity with normal polynomial expansions. Discussion Papers, ISDS 88-02, Duke University, Durham, 1987.

[55] J. Geweke. Acceleration methods for monte carlo integration in bayesian inference. Discussion Papers, ISDS 88-04, Duke University, Durham, 1988.

[56] R.G. Ghanem and P.D. Spanos. *Stochastic Finite Elements: A Spectral Approach.* Springer-Verlag, New York, 1991.

[57] M. Grigoriu. Tables of dimensionless central moments. *Journal of the Engineering Mechanics Division*, 106(EM6):1423–1429, 1980.

[58] M. Grigoriu. Methods for approximate reliability analysis. *Structural Safety*, 1:155–165, 1982.

[59] I.E. Grossmann and C.A. Floudas. Active constraint strategy for flexibility analysis in chemical processes. *Comput. Chem. Eng.*, 11(6):675–693, 1987.

[60] K.P. Halemane and I.E. Grossmann. Optimal process design under uncertainty. *AIChE Journal*, 29(3):425–433, 1983.

[61] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

[62] J.H. Halton. A retrospective and prospective survey of the monte carlo method. *SIAM Review*, 12(1):1–63, 1970.

[63] R. Hammer, M. Neaga, and D. Ratz. PASCAL-XSC, new concepts for scientific computation and numerical data processing. In E. Adams and U. Kulisch, editors, *Scientific Computing with Automatic Result Verification*, pages 15–44. Academic Press, San Diego, 1993.

[64] J.M. Hammersley and K.W. Morton. A new monte carlo technique antithetic variates. *Proc. Cambridge Phil. Soc.*, (52):449–474, 1956.

[65] J.C. Helton. Risk, uncertainty in risk, and the EPA release limits for radioactive waste disposal. *Nuclear Technology*, 101:18–39, 1993.

[66] T.W. Hill Jr. *On Determining a Distribution Function Known Only by Its Moments and/or Moment Generating Function.* Ph.D. dissertation, Arizona State University, 1969.

[67] A.C. Hindmarsh. ODEPACK, a systematized collection of ode solvers. In R.S. Stepleman, M. Carver, et al., editors, *Scientific Computing: Applications of Mathematics and Computing to the Physical Sciences*, pages 55–64. North-Holland Pub., 1983.

[68] T. Imamura, W.C. Meecham, and A. Siegel. Symbolic calculus of the wiener process and wiener-hermite functionals. *J. Math. Phys.*, 6(5):695–706, 1965.

[69] F. James. A review of pseudo-random number generators. *Computer Physics Comm.*, (60):329–344, 1990.

[70] E.T. Jaynes. Information theory and statistical mechanics. *Phys. Rev.*, 106:620–630, 1957.

[71] N.L. Johnson. Systems of frequency curves generated by methods of translation. *Biometrika*, 36:149–176, 1949.

[72] N.L. Johnson and S. Kotz. *Distributions in Statistics: Continuous Univariate Distributions-1.* Houghton Mifflin Co., Boston, 1970.

[73] N.L. Johnson and S. Kotz. *Distributions in Statistics: Continuous Multivariate Distributions.* John Wiley & Sons, Inc., New York, 1972.

[74] I.T. Jolliffe. *Principal Component Analysis.* Springer-Verlag, New York, 1986.

[75] S. Kakutani. Spectral analysis of stationary gaussian processes. In J. Neyman, editor, *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, volume II, pages 239–247. University of California Press, 1961.

[76] J.R. Kalagnanam and U.M. Diwekar. A comparison of optimal and latin hypercube sample points for uncertainty analysis. *submitted for publication.*

[77] H. Katzan. *Managing Uncertainty: A Pragmatic Approach.* Van Nostrand Reinhold, New York, 1992.

[78] A. Kaufmann and M.M. Gupta. *Introduction to Fuzzy Arithmetic: Theory and Applications.* Van Nostrand Reinhold Co., New York, 1985.

[79] E.H. Kerner. Universal formats for nonlinear ordinary differential equations. *J. Math. Physics*, 22(7):1366–1371, 1981.

[80] B.W. Kernighan and R. Pike. *The UNIX Programming Environment.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[81] M. Kleiber and T.D. Hien. *The Stochastic Finite Element Method: Basic Perturbation Technique and Computer Implementation.* John Wiley & Sons, New York, 1992.

[82] G.J. Klir and T.A. Folger. *Fuzzy Sets, Uncertainty, and Information.* Prentice Hall, Englewood Cliffs, 1988.

[83] D.E. Knuth. *The Art of Computer Programming.* Addison-Wesley, Reading, 1981.

[84] M.A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE J.*, 37(2):233–243, 1991.

[85] A. Kraslawski, T. Koiranen, and Nystrom L. Concurrent engineering: Robust design in fuzzy environment. *Computers and Chemical Engineering*, 17S:S447–S452, 1993.

[86] K. Krischer, R.R. Martinez, I.G. Kevrekidis, H.H. Rotermund, G. Ertl, and J.L. Hudson. Model identification of a spatiotemporally varying catalytic reaction. *AIChE J.*, 39(1):89–98, 1993.

[87] R. Kruse. On the semantic foundations of fuzzy probability theory and fuzzy statistics. In H. Bandemer, editor, *Modelling Uncertain Data*, pages 62–68. Akademie Verlag, Berlin, 1993.

[88] S. Kullback. *Information Theory and Statistics.* Wiley, New York, 1959.

[89] S. Kutscher and J. Schulze. Some aspects of uncertain modelling experiences in applying interval mathematics to practical problems. In H. Bandemer, editor, *Modelling Uncertain Data*, pages 62–68. Akademie Verlag, Berlin, 1993.

[90] S. Kwapien and W.A. Woyczynski. *Random Series and Stochastic Integrals: Single and Multiple.* Birkhauser, Boston, 1992.

[91] C. Lawo. C-XSC, a programming environment for verified scientific computing and numerical data processing. In E. Adams and U. Kulisch, editors, *Scientific Computing with Automatic Result Verification*, pages 71–86. Academic Press, San Diego, 1993.

[92] M.D. Lax. Approximate solution of random differential and integral equations. In G. Adomian, editor, *Applied Stochastic Processes*, pages 121–134, New York, 1980. Academic Press, Inc.

[93] Y.S. Lee and T.K. Lin. High order Cornish-Fisher expansion. *Appl. Statist.*, 41(1):233–240, 1992.

[94] T.G. Lewis and W.H. Payne. Generalized feedback shift register pseudorandom number algorithm. *Journal of the ACM*, 20(3):456–468, 1973.

[95] X. Li and R.S. Tankin. On the prediction of droplet size and velocity distributions in sprays through maximum entropy principle. *Part. Part. Syst. Charact.*, 9:195–201, 1992.

[96] J.-S. Liou, V. Balakotaiah, and D. Luss. Dispersion and diffusion influences on yield in complex reaction networks. *AIChE Journal*, 35:1509–1520, 1989.

[97] E. Lukacs. *Characteristic Functions.* Griffin, London, 1970.

[98] J.R. Lund. Random variables versus uncertain values: Stochastic modeling and design. *Journal of Water Resources Planning and Management*, 117(2):179–194, 1991.

[99] A. Madansky. Some results and problems in stochastic linear programming. Tech. report, RAND Corporation, 1959.

[100] V.A. Malyshev and R.A. Minlos. *Gibbs Random Fields: Cluster Expansions.* Kluwer Academic Publishers, Dordrecht, 1991.

[101] V. Manousiouthakis and D. Sourlas. A global optimization approach to rationally constrained rational programming. *Chem. Eng. Comm.*, 115:127–147, 1992.

[102] G. Marsaglia, Narasimhan B., and A. Zaman. A random number generator for PC's. *Computer Physics Communications*, 60:345–349, 1990.

[103] G. Marsaglia and T.A. Bray. A convenient method for generating normal variables. *SIAM Rev.*, 6:260–264, 1964.

[104] A.M. Mathai and R.K. Saxena. *The H-Function with Applications in Statistics and Other Disciplines.* John Wiley & Sons, New York, 1978.

[105] W.L. McCabe, J.C. Smith, and P. Harriott. *Unit Operations of Chemical Engineering.* McGraw-Hill Publishing Co., 1985.

[106] R.L. McCreanor, S.S. Udpa, and L. Udpa. Interpolation of images via moments. *International Journal of Modeling & Simulation*, 12(4):149–153, 1992.

[107] E.D. McCune and H.L. Gray. Cornish-Fisher and Edgeworth expansions. In S. Kotz and N.L. Johnson, editors, *Encyclopedia of Statistical Sciences*, pages 188–193. John Wiley & Sons, 1982.

[108] C. McGeoch. Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups. *ACM Computing Surveys*, 24(2):195–212, 1992.

[109] U.B. Mehta. Some aspects of uncertainty in computational fluid dynamics results. *Journal of Fluids Engineering*, 113(4):538–543, 1991.

[110] R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, 1966.

[111] R.E. Moore. Parameter sets for bounded-error data. *Mathematics and Computers in Simulation*, 34:113–119, 1992.

[112] M.G. Morgan, M. Henrion, and M. Small. *Uncertainty, A Guide to Dealing with Uncertainty in Quantitative Risk and Policy Analysis.* Cambridge University Press, New York, 1992.

[113] U.G. Näf. Stochastic simulation using gPROMS. Technical report, Imperial College, London, 1992.

[114] G.T. Nicol. *Flex, the Lexical Scanner Generator.* Free Software Foundation, Cambridge, Massachusetts, 1993.

[115] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM, Philadelphia, 1992.

[116] E.M. Oblow. O-theory: A probabilistic alternative to fuzzy set theory. In B. Bouchon and R.R. Yager, editors, *Uncertainty in Knowledge-Based Systems*, pages 111–119. Springer-Verlag, Berlin, 1987.

[117] M.K. Ochi. Non-gaussian random processes in ocean engineering. *Probabilistic Engineering Mechanics*, 1(1):28–39, 1986.

[118] J.K. Ord. *Families of Frequency Distributions.* Griffin Co., London, 1972.

[119] G. Palm and T. Poggio. The volterra representation and the wiener expansion: Validity and pitfalls. *Siam J. Appl. Math.*, 33(2):195–216, 1977.

[120] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, 1991.

[121] J. Pearl. Rejoinder to comments on "reasoning with belief functions: An analysis of compatibility". *International Journal of Approximate Reasoning*, 6(3):425–443, 1992.

[122] K Pearson. Contributions to the mathematical theory of evolution. ii. skew variations in homogeneous material. *Philosophical Transactions of the Royal Society of London, Series A*, 186:343–414, 1895.

[123] J.E. Penner, R.J. Charlson, J.M. Hales, N.S. Laulainen, R. Leifer, T. Novakov, J. Ogren, L.F. Radke, S.E. Schwartz, and L. Travis. Quantifying and minimizing uncertainty of climate forcing by anthropogenic aerosols. *Bulletion of the American Meteorological Society*, 75(3):375–400, 1994.

[124] E.N. Pistikopoulos and Mazzuchi T.A. A novel flexibility analysis approach for processes with stochastic parameters. *Computers Chem. Engng.*, 14(9):991–1000, 1990.

[125] G.J. Powers. Design of a catalytic reactor-separator system with uncertainty in catalyst activity and selectivity. *Ind. Eng. Chem. Process Des. Develop.*, 14(1):41–44, 1975.

[126] R.D. Prasad. Probability distributions of algebraic functions of independent random variables. *SIAM J. Appl. Math.*, 18(3):614–626, 1970.

[127] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, New York, 1992.

[128] W.H. Press and S.A. Teukolsky. Padé approximants. *Computers in Physics*, 6(1):82–83, 1992.

[129] G. M. Provan. The validity of Dempster-Shafer belief functions. *International Journal of Approximate Reasoning*, 6(3):389–399, 1992.

[130] V.S. Pugachev. *Probability Theory and Mathematical Statistics for Engineers.* Pergamon Press, 1984.

[131] P. Quinio and T. Matsuyama. Random closed sets: A unified approach to the representation of imprecision and uncertainty. In R. Kruse and P. Siegel, editors, *Symbolic and Quantitative Approaches to Uncertainty*, pages 282–286. Springer-Verlag, Berlin, 1991.

[132] D. Ramkrishna. On problem-specific polynomials. *Chemical Engineering Science*, 28:1362–1365, 1973.

[133] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Ellis Horwood, New York, 1984.

[134] G. Revesz. *Lambda-Calculus Combinators and Functional Programming.* Cambridge University Press, Cambridge, 1988.

[135] C. Ritter and M.A. Tanner. Facilitating the gibbs sampler: The gibbs stopper and the griddy-gibbs sampler. *J. Amer. Stat. Assoc.*, 87(419):861–868, 1992.

[136] T. Ritter. The efficient generation of cryptographic confusion sequences. *Cryptologia*, 15(2):81–139, 1991.

[137] G. Rowlands and J.C. Sprott. Extraction of dynamical equations from chaotic data. *Physica D*, 58:251–259, 1992.

[138] R.Y. Rubinstein. *Simulation and the Monte Carlo Method.* John Wiley & Sons, New York, 1981.

[139] H. Sarper. Monte carlo simulation for analysis of the optimum value distribution in stochastic mathematical programs. *Mathematics and Computers in Simulation*, 35:469–480, 1993.

[140] H.H. Seinfeld. *Atmospheric Chemistry and Physics of Air Pollution.* John Wiley & Sons, New York, 1986.

[141] F.M. Selten. Toward an optimal description of atmospheric flow. *J. Atmos. Sci.*, 50(6):861–877, 1993.

[142] S.E. Serrano. Using the c language to approximate non-linear stochastic systems. *Adv. Eng. Software*, 12(2):59–68, 1990.

[143] S.E. Serrano and T.E. Unny. Random evolution equations in hydrology. *Applied Mathematics and Computation*, 39:97s–122s, 1990.

[144] H.E. Shaalan and R.P. Broadwater. Using interval mathematics in cost-benefit analysis of distribution automation. *Electric Power Systems Research*, 27:145–152, 1993.

[145] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, 1976.

[146] G. Shafer. Rejoinder to comments on "perspective on the theory and practice of belief functions". *International Journal of Approximate Reasoning*, 6(3):445–480, 1992.

[147] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423; 623–656, 1948.

[148] Y.S. Sherif and D.L. Minh. Improving the statistical efficiency of computer simulation experiments. *Microelectron. Reliab.*, 33(3):373–384, 1993.

[149] J.E. Shore and R.W. Johnson. Properties of cross-entropy minimization. *IEEE Trans. Info. Theory*, IT-27(4):472–482, 1981.

[150] W.T. Silva and R.L. Milidiú. Algorithm for combining belief functions. *International Journal of Approximate Reasoning*, 7(1,2):73–94, 1992.

[151] L. Sirovich and R. Everson. Management and analysis of large scientific datasets. *Int. J. Supercomp. Appl.*, 6(1):50–68, 1992.

[152] S. Skogestad, M. Morari, and J.C. Doyle. Robust control of ill-conditioned plants: High-purity distillation. *IEEE Transactions on Automatic Control*, 33(12):1092–1105, 1988.

[153] A.F.M Smith. Bayesian computational methods. *Phil. Trans. R. Soc. Lond.*, 337 A:369–386, 1991.

[154] A.F.M. Smith and A.E. Gelfand. Bayesian statistics without tears: A sampling-resampling perspective. *Amer. Statistician*, 46(2):84–88, 1992.

[155] K. Sobczyk and J. Trębicki. Maximum entropy principle and nonlinear stochastic oscillators. *Physica A*, 193:448–468, 1993.

[156] P.N. Souza. Computer algebra systems. *Notices of the American Mathematical Society*, 40(6):617–623, 1993.

[157] D.J. Spiegelhalter. A statistical view of uncertainty in expert systems. In W.A. Gale, editor, *Artificial Intelligence and Statistics*, pages 17–55. Addison-Wesley, Reading, 1986.

[158] M.D. Springer. *The Algebra of Random Variables.* John Wiley & Sons, New York, 1979.

[159] H.M. Srivastava, K.C. Gupta, and S.P. Goyal. *The H-Functions of One and Two Variables with Applications.* South Asian Publisher, New Delhi, 1982.

[160] I.M. Stancu-Minasian. *Stochastic Programming with Multiple Objective Functions.* D. Reidel Publishing Company, Boston, 1984.

[161] W.E. Stewart, M. Caracotsios, and J.P. Sørensen. Parameter estimation from multiresponse data. *AIChE Journal*, 38(5):641–650, 1992.

[162] H.S. Steyn. On regression properties of multivariate probability functions of pearson's types. *Proceedings of the Royal Academy of Sciences, Amsterdam*, 63:302–311, 1960.

[163] D.A. Straub and I.E. Grossmann. Integrated stochastic metric of flexibility for systems with discrete state and continuous parameter uncertainties. *Computers Chem. Engng.*, 14(9):967–985, 1990.

[164] M.A. Tatang. Combined stochastic and deterministic approach in solving stochastic differential equations. M.S. thesis, Carnegie Mellon University, 1992.

[165] M.A. Tatang and G.J. McRae. A new approach for calculating flexibility index. *in preparation.*

[166] M.A. Tatang and G.J. McRae. A new approach for addressing chemical engineering design and optimization problems in the presence of uncertainty. AIChE Annual Meeting, November 1993.

[167] B. Tessem. Interval probability propagation. *International Journal of Approximate Reasoning*, 7(3,4):95–120, 1992.

[168] W.E. Thompson and P.A. Palicio. Bayesian confidence limits for the availability of systems. *IEEE Transactions on Reliability*, R-24:118–120, 1975.

[169] W.D. Tian. Cumulant based probabilistic power system simulation using laguerre polynomials. *IEEE Trans. Energy Conv.*, 4(4):567–573, 1989.

[170] G. Tintner. Stochastic linear programming with applications to agricultural economics. In H.A. Antonosiewicz, editor, *Proceedings of the Second Symposium in Linear Programming*, pages 197–228, Washington, 1955. National Bureau Standard.

[171] A.D. Trim. Estimation of density functions and upcrossing-rates from samples. *Probabilistic Engineering Mechanics*, 4(4):191–202, 1989.

[172] A.A. Trukhachev. Application of orthogonal expansions of probability distribution densities. *Soviet Journal of Communications Technology & Electronics*, 36(8):19–22, 1991.

[173] Y.K. Tung. Mellin transforms applied to uncertainty analysis in hydrology / hydraulics. *Journal of Hydraulic Engineering*, 116(5):659–674, 1990.

[174] S.M. Ulam. *Adventures of a Mathematician.* Scribner, New York, 1976.

[175] L.T. Urgell and R.L. Kirlin. Adaptive image compression using Karhunen-Loève transform. *Signal Processing*, 21:303–313, 1990.

[176] S. Vajda. Stochastic programming. In J. Abadie, editor, *Integer and Nonlinear Programming*, chapter 14, pages 321–336. North-Holland Publishing Company, Amsterdam, 1970.

[177] S. Vajda. *Probabilistic Programming.* Academic Press, New York, 1972.

[178] E. Vanmarcke. *Random Fields: Analysis and Synthesis.* The MIT Press, Cambridge, 1983.

[179] V.M. Veliov. Parametric and functional uncertainties in dynamic systems: Local and global relationship. In L. Atanassova and J. Herzberger, editors, *Computer Arithmetic and Enclosure Methods*, pages 427–436. Elsevier Science Publishers, Amsterdam, 1992.

[180] J.A.M. Vermaseren. *FORM*, 1989.

[181] J.A.M. Vermaseren. FORM, a program for the manipulation of very large formulae. *A part of FORM software documentation*, pages 1–13, 1989.

[182] R. Viertl. On statistical inference based on non-precise data. In H. Bandemer, editor, *Modelling Uncertain Data*, pages 62–68. Akademie Verlag, Berlin, 1993.

[183] J. Villadsen and M.L. Michelsen. *Solution of Differential Equation Models by Polynomial Approximation.* Prentice-Hall Inc., Englewood Cliffs, 1978.

[184] R. Von Mises. *A Mathematical Theory of Probability and Statistics.* Academic Press, New York, 1964.

[185] J. Von Neumann. Various techniques used in connection with random digits. *U.S. Nat. Bur. Stand. Appl. Math. Ser.*, 12:36–38, 1951.

[186] P. Walley. *Statistical Reasoning with Imprecise Probabilities.* Chapman and Hall, London, 1991.

[187] W.V. Walter. ACRITH-XSC, a Fortran-like language for verified scientific computing. In E. Adams and U. Kulisch, editors, *Scientific Computing with Automatic Result Verification*, pages 45–70. Academic Press, San Diego, 1993.

[188] T. Watanabe and H. Ellis. Stochastic programming models for air quality management. *Computers Ops. Res.*, 20(6):651–663, 1993.

[189] R.J.B. Wets. Stochastic programs with fixed recourse: The equivalent deterministic program. *SIAM Review*, 16(3):309–339, 1974.

[190] N. Wiener. The homogeneous chaos. *Amer. J. Math.*, 60:897–936, 1938.

[191] K.-D. Willamowski and O.E. Rössler. Irregular oscillations in a realistic abstract quadratic mass action system. *Z. Naturforsch.*, 35a:317–318, 1980.

[192] K.L. Wood, K.N. Otto, and E.K. Antonsson. Engineering design calculations with fuzzy parameters. *Fuzzy Sets and Systems*, 52:1–20, 1992.

[193] H. Wozniakowski. Average case complexity of multivariate integration. *Bull. Amer. Math. Soc.*, 24(1):185–194, 1991.

[194] Y.T. Wu and P.H. Wirsching. New algorithm for structural reliability estimation. *Journal of Engineering Mechanics*, 113(9):1319–1336, 1987.

[195] R.R. Yager. Arithmetic and other operations on dempster-shafer structures. *Int. J. Man-Machine Studies*, 25:357–366, 1986.

[196] H.Q. Yang, H. Yao, and J.D. Jones. Calculating functions of fuzzy numbers. *Fuzzy Sets and Systems*, 55:273–283, 1993.

[197] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[198] L.A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428, 1975.

[199] A. Zellner. Optimal information processing and bayes' theorem. *The American Statistician*, 42(4):278–280, 1988.

[200] D. Zwillinger. *Handbook of Integration.* Jones and Bartlett, Boston, 1992.