

**Uncertainty operations with Statool**

by

**Jianzhong Zhang**

A thesis submitted to the graduate faculty  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE

Major: Computing Engineering

Program of Study Committee:  
Daniel Berleant, Major Professor  
Gerald Sheblé  
Soumendra Nath Lahiri

Iowa State University

Ames, Iowa

2002

Copyright © Jianzhong Zhang, 2002. All rights reserved.

Graduate College  
Iowa State University

This is to certify that the master's thesis of

Jianzhong Zhang

has met the thesis requirements of Iowa State University

---

Major Professor

---

For the Major Program

## Table of Contents

List of Figures .....	v
List of Tables .....	vii
Acknowledgements .....	viii
Abstract .....	ix
1 Introduction .....	1
1.1 Interval mathematics .....	3
1.2 Interval-based analysis .....	5
2 Narrowing the envelopes around results using correlation .....	11
2.1 Facts about correlation .....	11
2.2 Joint distributions .....	12
2.3 Interval-valued correlations .....	12
2.4 Legal and illegal correlation values .....	14
2.4.1 Solution .....	14
2.4.2 Approximate solution .....	16
2.5 Additional constraints gotten from correlation .....	16
2.6 Nonlinear optimization to remove excess width .....	17
2.7 Improving results by adding constraints to LP .....	18
2.8 Improved simplex method .....	19
2.8.1 How to find the initial feasible solution .....	21
2.8.2 How to decide the termination condition and entering variable .....	22
2.8.3 How to determine the leaving variable .....	23
2.8.4 Decreasing computing .....	24
2.8.5 Applying method .....	25
2.8.6 An example .....	26
2.9 Nonlinear optimization .....	27
2.9.1 Local and global optimums .....	28
2.9.2 Classical theory of unconstrained optimization .....	29
2.9.3 Finding a solution iteratively .....	29
2.9.4 Search methods: direct and gradient .....	30
2.9.5 Converting constrained to unconstrained optimization .....	30
2.9.6 Our case .....	32
3 Enhancement of functions .....	36
3.1 Transportation method .....	36
3.1.1 Background on the transportation simplex method .....	36
3.1.2 Exceptions in finding the initial solution .....	39
3.1.3 Adaptation to the unknown dependency case .....	39
3.1.4 Test result: .....	41
3.2 Cascading operations .....	42
3.2.1 Solution .....	43
3.3 Relational operations .....	45
3.3.1 Relational operations on intervals .....	45
3.3.2 Relational operations on random variables .....	45
3.4 Complex expressions .....	47

3.4.1	Expression editor .....	47
3.4.2	Limitations on evaluating expressions.....	48
3.4.3	Excess width in expressions.....	49
4	Software architecture .....	51
4.1	Overview .....	51
4.2	Input/output.....	52
4.3	The user interface.....	53
4.3.1	Main primary window: the operation window.....	54
4.3.2	Other primary windows: the data editor and the view windows .....	56
4.3.3	Windows management.....	57
4.4	The logical layer .....	58
4.5	The computing layer .....	59
5	Component design and implementation.....	61
5.1	Overview.....	61
5.2	Operation and properties windows .....	61
5.2.1	Operation window.....	61
5.2.2	Properties windows of the operation window.....	68
5.3	Other primary windows .....	70
5.4	Lower 2 levels of logical layer.....	70
5.4.1	Utility functions and internal data structure.....	71
5.4.2	Access points to the computing layer .....	71
5.5	The computing layer .....	73
5.5.1	Converting data.....	73
5.5.2	General (legacy) simplex method .....	75
5.5.3	Transportation simplex method .....	76
5.5.4	Incorporating correlation as a constraint.....	78
6	Explanation of the results.....	92
6.1	Experiments .....	92
6.2	Discussion .....	97
	References.....	98

## List of Figures

Figure 1.1. Probability bounds.....	10
Figure 3.1. Convert CDF to IDF.....	43
Figure 3.2. Result for operation.....	43
Figure 3.3. Result for $x+y$ .....	44
Figure 3.4. Result for $x+y+z$ .....	44
Figure 3.5. Expression editor.....	48
Figure 3.6. Error information for experssion editor.....	48
Figure 4.1. Architecture.....	51
Figure 4.2. Operation window.....	55
Figure 4.3. Data editor.....	56
Figure 4.4. Data view.....	57
Figure 4.5. Logical layer.....	59
Figure 4.6. Package view for the computing layer.....	60
Figure 5.1. Operation window design.....	62
Figure 5.2. File menu.....	62
Figure 5.3. Edit menu.....	63
Figure 5.4. Switch menu.....	63
Figure 5.5. View menu.....	64
Figure 5.6. Options menu.....	64
Figure 5.7. Help menu.....	64
Figure 5.8. Popup menu for operand X.....	65
Figure 5.9. Popup menu for operand Y.....	65
Figure 5.10. Popup menu for result Z.....	65
Figure 5.11. Correlation setting.....	66
Figure 5.12. Operation types.....	67
Figure 5.13. Display mode window.....	68
Figure 5.14. Display color window.....	69
Figure 5.15. Correlation setting window.....	69
Figure 5.16. Expression editor window.....	70
Figure 5.17. Class relationship for transportation simplex method.....	77
Figure 5.18. Class CConfigure.....	77
Figure 5.19. Flow of control for transportation Simplex implementation.....	78
Figure 5.20. Class: COptimalMin.....	81
Figure 5.21. Class: CConfigure.....	82
Figure 5.22. Class: CVariance.....	82
Figure 5.23. Class: CSimplex.....	83
Figure 5.24. Class CMaxmin.....	84
Figure 5.25. Class: CBoundCorrelation.....	85
Figure 5.26. Class: CMinCorrelation.....	86
Figure 5.27. Class: CMinCorrelationExy.....	87
Figure 5.28. Class diagram.....	87
Figure 5.29. Sequence diagram for Cor_Bound.....	88

Figure 5.30. Sequence diagram for Cor_Min. ....	89
Figure 5.31. Sequence diagram for Cor_Min_Exy. ....	90
Figure 5.32. Sequence diagram for Cor_Min_Exy_S. ....	91
Figure 6.1. $X+Y$ when $X$ and $Y$ are 16 intervals. ....	92
Figure 6.2. $X+Y$ when $X$ and $Y$ are 32 intervals. ....	93
Figure 6.3. $X+Y$ when $X$ and $Y$ have 64 intervals. ....	93
Figure 6.4. $X*Y$ for unknown. ....	94
Figure 6.5. $X*Y$ for correlation 0.98. ....	94
Figure 6.6. $X*Y$ for correlation 0. ....	94
Figure 6.7. $X*Y$ for correlation -0.98. ....	95
Figure 6.8. Times for operations. ....	96
Figure 6.9. Times for multiplication and division. ....	96

## List of Tables

Table 1.1. Distributions for X and Y. ....	6
Table 1.2. Marginal distribution for X and Y. ....	6
Table 1.3. Joint distribution for independency. ....	7
Table 1.4. Joint distribution for specified value. ....	7
Table 1.5. Probabilities for result variable. ....	10
Table 2.1. Joint distribution matrix. ....	15
Table 2.2. Joint distribution for X and Y. ....	18
Table 3.1. Parameter table for transporation model. ....	37
Table 3.2. Marginal distribution ....	40
Table 3.3. Lower bound. ....	42
Table 3.4. Upper bound. ....	42
Table 3.5. Distribution for X and Y. ....	46
Table 3.6. Interval value for relational operation. ....	46
Table 6.1. Operation evaluation time (seconds) for correlation 0. ....	95

## Acknowledgements

To finish this thesis in English was a challenging experience. I am glad that I did it. I am grateful to many people.

First to my major professor, Dr. Daniel Berleant. I want to thank him for giving me the chance to work with him closely on the research project and for his great advice on my graduate studies in these two years. Dr. Berleant read the drafts of my thesis and gave me a lot of constructive feedback even when he was in vacation with his family in the summer. I was always moved when I got his comments and corrections on my thesis via email. I appreciate deeply his time and work. It has been a very good luck for me to have worked with him. What I have learned from him will benefit me in my all life.

I also want to thank Dr. Gerad Sheble with whom I have worked on the research project. Our weekly meetings have been great opportunities for me to learn from him. I have enjoyed the meetings. Thanks also to my friends who give me support and understanding.

I am heavily indebted to my parents-in-law. They took good care of my newborn baby and my wife when I was working on the final drafts of my thesis. I thank them for freeing me from the household chores and the moral support they gave me.

I want to say “thank you” to my dear parents for instilling in me the value of endless learning and a thirst for knowledge. Without it, I would not have even been able to achieve this degree.

Finally my heartfelt thanks go to my wife, De, and to my baby daughter, Peilan for their wonderful support, and for not minding the many lost evenings and weekends during my preparation for this thesis.



## Abstract

Uncertainty exists frequently in our knowledge of the real world. Probability is a common way to measure uncertainty. People sometimes define random variables whose values are derived from arithmetic operations on other random variables. Generally there are two classes of methods to handle this topic: analytical and numerical. Analytical methods are restricted to specific classes of input distributions. Numerical methods only give numerical results and are widely used in real applications if approximate results can be accepted.

Monte Carlo simulation is one of the best-known numerical methods. However the traditional approach of Monte Carlo has some limitations. Interval-based dependency analysis (DEnv) was developed by Berleant and Goodman-Strauss. Another approach is the copula-based approach. These two methods have been implemented in software. The copula-based approach is implemented in the commercial software RiskCalc. DEnv is implemented in Statool.

The current Statool supports a variety of dependence relationships: independence, unknown dependence, and specific correlation values. The algorithm extension to support correlation is a significant improvement. The current version of Statool uses the transportation simplex method to speed up computing. Cascaded operations, relational operations and monotonic binary functions are newly supported by the current Statool. These new functions, and using correlation as constraints, are the main advances in Statool.

This software is based on a layer design including the user interface, the logical layer, and the computing layer. This is suitable for implementation and maintenance of distributed and other computing software. OO methods are adapted. Unified Modeling Language (UML) gives visualization and documentation support for the computing layer. The main algorithms are implemented in many objects. Currently it is developed for a Microsoft Windows platform. Visual C++ and Visual Basic were used for development. Dynamically linked libraries are used to contain the components of the computing layer.

# 1 Introduction

Uncertainty exists frequently in our knowledge of the real world. Handling uncertainty is therefore a common problem. Probability is a common way to measure the level of uncertainty. Probability density functions (PDF) or their integrals, cumulative distribution functions (CDF), are often used to model the uncertainty in the value of a quantity. Often, uncertainty can be stated by using a random variable. But this is not enough. People some times define random variables, whose sample values are derived from arithmetic operations on the values of other random variables.

Binary operations are very basic and common operations. When two random variables are operated on to derive a new random variable, the distribution to describe this random variable is termed a derived distribution (Springer, 1979). Such operations are well recognized in many fields, such as decision analysis and risk analysis, and many other fields as well.

A variety of methods have been developed to address this topic. Generally there are two classes of methods to handle it: analytical and numerical. Analytical methods are restricted to specific classes of input distribution, under assumptions, such as independence. For example, normal distributions are often used. If two random variables are normal and independent, the sum of these two random variables still is normal. It is also possible to obtain derived distributions for specified dependency relationships other than independence, such as perfect positive rank correlation. However, it is often not easy to find analytical results for random variable operations and it is not always reasonable to make convenient assumptions about dependency. Sometimes, we don't have any information about dependency. But an advantage of analytical methods is accurate result. Unlike analytical methods, numerical methods only give numerical results. But this is suitable for a wide class of distributions. Numerical methods are widely used in real applications if approximate results can be accepted within specific tolerances.

Monte Carlo simulation is one of the best-known numerical methods. However, the traditional approach of Monte Carlo has some limitations. It assumes the distribution of the random variables is known, and their relationship is independent or known (Ferson 1996). If either the probability distributions or the dependency relationship of the random variables are

not available, some assumptions are usually made to process it. If the assumptions don't hold, results can be seriously affected.

A discretized convolution approach can be used to calculate the result for the independent situation (Ingram et al. 1968; Colombo and Jaarsma 1980; Kaplan 1981). Interval analysis can be used to solve this problem. (It is obvious that interval numbers will be really close to point values if the interval is narrow enough.) Interval mathematics can then be applied (Moore, 1966).

Intervals have the potential for bounding the result of an operation. Discretization error coming from discretizing distributions may be bounded by interval based discretization (Berleant 1993). If the dependency is not specified, result bounds will include the entire range of possible dependencies. These bounds should be wider than if a particular dependency is specified. Interval-based dependency analysis is developed by Berleant and Goodman-Strauss (1998). This approach has fundamental similarities with the copula-based approach (Frank et al. 1987), which was significantly extended by Williamson and Downs (1990). These two methods have been implemented in software. The copula-based approach, termed probabilistic arithmetic, is implemented in the commercial software RiskCalc (Ferson et al. 1998). DEnv is implemented as Statool (Berleant and Goodman-Strauss 1998), which extends the previous tool (Berleant and Cheng 1998) through eliminating the independence assumption. Statool can handle the case where a dependency relationship is unknown or unspecified, by not making any assumption about the dependency relationship between operands. But partial dependence information might be available in some cases. If we can use this information in the calculation, we will get more accurate results than can be obtained without using this information.

The current Statool supports a variety of dependence relationships, such as independence, unknown dependence, and correlation values. The algorithm extension to support correlation is a significant improvement. The current version of Statool uses the transportation simplex method to speed up computing. Cascaded operations and monotonic binary functions are supported by the current Statool. These new functions, and using correlation as a constraint, are the main advances in Statool. Among the other contributions reported here are addressing example application problems.

## 1.1 Interval mathematics

Interval mathematics was developed by Moore (1966). Compared with the real domain, let us see what interval arithmetic and analysis are.

An interval value is composed of 2 real numbers, which are called the low bound and high bound. For example, given interval value  $X=[a,b]$ ,  $a$  and  $b$  are real numbers, and  $a$  is the low bound and  $b$  is the high bound. Thus, we can see that an interval value in the interval system corresponds to an interval in the real system. If  $a$  is equal to  $b$ , this interval value is the real number  $a$ . Or you can use set theory to describe the interval  $X=[a,b]$ . We can define it as a set  $X=\{x: a \leq x \leq b\}$ . Next, we will define how to describe the relationship between two interval values. If we say  $[a,b]=[c,d]$ , it means  $a=c$  and  $b=d$ . if  $[a,b]<[c,d]$ , it means  $b<c$ .

Interval arithmetic includes addition, subtraction, multiplication and division. Here  $X=[a,b]$  and  $Y=[c,d]$  are two intervals. The following gives the definition for arithmetic based on the set definition for intervals.

$$X \otimes Y = \{x \otimes y : x \in X, y \in Y\}$$

where  $\otimes$  is in  $+, -, *, /$ .

Clearly,  $X+Y = [a+c, b+d]$  and  $X-Y = [a-d, b-c]$ . Multiplication is a little more complex.

$X*Y = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$ . And division is even more complex. First, note that  $Y$  doesn't include zero.

$$1/Y = [1/d, 1/c] \text{ if } 0 \notin Y$$

$$X/Y = X * (1/Y) \text{ if } 0 \notin Y$$

If  $Y$  is an interval including zero,  $X/Y$  should be  $[-\infty, \infty]$  if the interval system includes infinities as allowable endpoints.

Interval arithmetic also includes the following characteristics:

- Set Rule
  - $(V \cup W) \pm Z = (V \pm Z) \cup (W \pm Z)$
- Rule for the addition and subtraction of infinite or semi-infinite intervals
  - $[a,b] + [-\infty, d] = [-\infty, b+d]$
  - $[a,b] + [c, \infty] = [a+c, \infty]$
  - $[a,b] \pm [-\infty, \infty] = [-\infty, \infty]$
  - $[a,b] - [-\infty, d] = [a-d, \infty]$

- $[a,b]-[c,\infty] = [-\infty,b-c]$
- Associativity and Commutativity
  - $X+(Y+Z) = (X+Y)+Z$
  - $X*(Y*Z) = (X*Y)*Z$
  - $X+Y = Y+X$
  - $X*Y = Y*X$

Unlike in real arithmetic, operations are not invertible, which means there is no inverse operation existing for a given operation. For the real domain, we know  $+$  and  $-$  are inverse operations, but in interval mathematics, this is not true.

In interval analysis, interval functions form a major topic. An interval function  $F$  is an interval-valued function of one or more interval arguments. For a real-valued function  $f$  of real variables  $x_1, \dots, x_n$ , if we have an interval function  $F$  of interval variables  $X_1, \dots, X_n$ , and if  $F(x_1, \dots, x_n) = f(x_1, \dots, x_n)$  for all  $x_i (i=1, \dots, n)$  then  $F$  is an interval extension of  $f$ . Interval functions have the following characteristics:

Inclusion monotonicity

- If  $X_i \subseteq Y_i (i=1, \dots, n)$  then  $F(X_1, \dots, X_n) \subseteq F(Y_1, \dots, Y_n)$ .

Arithmetic inclusion monotonicity

- If  $op$  denotes  $+, -, *, /$ , then  $X_i \subset Y_i (i=1, 2)$  implies  $(X_1 op X_2) \subset (Y_1 op Y_2)$

Excess width is a big problem in interval mathematics. Let us use a simple example to explain this problem. For interval value  $X=[a,c]$ , what is the result for  $X-X$ ? In naïve interval arithmetic, the result is not zero, but  $[a-c, c-a]$ . Zero is just one real number included in this result. Obviously it is really not our expected result since it is too wide. For functions, an interval function extension need not be unique, but can depend on the form of the real function. For example, there may be three expressions corresponding to the same real function:

- $f_1(x) = x*x - x + 1$ ,  $f_2(x) = (x-1/2)^2 + 3/4$ , and  $f_3(x) = x*(x-1) + 1$ .
- The corresponding interval extensions are:
- $f_1(X) = X*X - X + 1$ ,  $f_2(X) = (X-1/2)^2 + 3/4$ , and  $f_3(X) = X*(X-1) + 1$ .

- These don't represent the same interval function, as:
- $f_1([0,2]) = [-1,5]$ ,  $f_2([0,2]) = [3/4,3]$ , and  $f_3([0,2]) = [-1,3]$ .
- The true range of  $f([0,2])$  is  $[3/4,3]$  computed by the interval function  $f_2$ , because  $x$  appears only once.
- This is referred to as the dependency problem or excess width.
- It enlarges intervals in the result collection.

The reason why excess width occurs is that a variable occurs more than one time in expression. So far, many methods have been developed to address this issue. Some methods are as follows.

- Various centered forms:
  - Computing the range of values (Asaithambi, Zuhe, and Moore, 1982)
  - Enclosure methods (Alefeld, 1990)
  - Artificial intelligence work (Hyvonen, 1992)

Computation time tends to be a problem with these excess width removal techniques. To apply interval analysis, the following guiding principles should be considered. (Walster 1998):

- “Interval algorithms should bound error”
- “Interval input/output conventions should be consistent with people's normal interpretation of numerical accuracy”
- “The application of interval algorithms should be universal”
- “Where interval algorithms currently do not exist, we should get to work developing them rather than abandoning the principle of universal applicability”

## **1.2 Interval-based analysis**

An interval can be used to bound the range for a value. If this interval is associated with a specified probability, as when the domain of a random variable is partitioned, we have lost information about probability distribution in this interval. The partitioning of the domain of a random variable into intervals and probabilities is the basis for extending binary operations from intervals to distributions.

At this point, we only consider the binary operations. We can extend binary operations and later we will talk about how to do this. Assuming there are 2 random variables  $X$  and  $Y$ , to get the exact distribution for the result of operation, we must know the joint distribution for random variables  $X$  and  $Y$ . The joint distribution is related to the correlation for these two random variables. Let us see an example.

Consider two random variables  $X$  and  $Y$ . This table shows their distributions.

**Table 1.1. Distributions for  $X$  and  $Y$ .**

	$X$			$Y$		
Range	[1,2]	[2,3]	[3,4]	[2,3]	[3,4]	[4,5]
Probability	0.25	0.5	0.25	0.5	0.3	0.2

We don't have any information about distribution within these ranges. And we also don't have any information about the dependency relationship between  $X$  and  $Y$ . Obviously, we don't know the joint distribution for  $X$  and  $Y$ .

Consider addition:  $Z=X+Y$ . Because we don't have the joint distribution for  $X$  and  $Y$ , it is impossible to find the exact result for  $Z$ . Now we put these two random variables into a matrix shown in the following table.

**Table 1.2. Marginal distribution for  $X$  and  $Y$ .**

$z \in [3,5]$	$z \in [4,6]$	$z \in [5,7]$	$y \in [2,3]$
$p_{11} = ?$	$p_{12} = ?$	$p_{13} = ?$	$p_{Y1} = 0.5$
$z \in [4,6]$	$z \in [5,7]$	$z \in [6,8]$	$y \in [3,4]$
$p_{21} = ?$	$p_{22} = ?$	$p_{23} = ?$	$p_{Y2} = 0.3$
$z \in [5,7]$	$z \in [6,8]$	$z \in [7,9]$	$y \in [4,5]$
$p_{31} = ?$	$p_{32} = ?$	$p_{33} = ?$	$p_{Y3} = 0.2$
$x \in [1,2]$	$x \in [2,3]$	$x \in [3,4]$	$\leftrightarrow \quad \updownarrow$
$p_{X1} = 0.25$	$p_{X2} = 0.5$	$p_{X3} = 0.25$	$X \quad Y$

The last row in the table is the distribution for  $X$  and last column is the distribution for  $Y$ . We don't know the value for cells  $p_{11}$  through  $p_{33}$  because we don't know the joint

distribution. For the simple case, if X and Y are independent, we can fill in the missing values as in the following table.

**Table 1.3. Joint distribution for independency.**

$z \in [3,5]$ $p_{11} = 0.125$	$z \in [4,6]$ $p_{12} = 0.25$	$z \in [5,7]$ $p_{13} = 0.125$	$y \in [2,3]$ $p_{Y1} = 0.5$
$z \in [4,6]$ $p_{21} = 0.075$	$z \in [5,7]$ $p_{22} = 0.15$	$z \in [6,8]$ $p_{23} = 0.075$	$y \in [3,4]$ $p_{Y2} = 0.3$
$z \in [5,7]$ $p_{31} = 0.05$	$z \in [6,8]$ $p_{32} = 0.1$	$z \in [7,9]$ $p_{33} = 0.05$	$y \in [4,5]$ $p_{Y3} = 0.2$
$x \in [1,2]$ $p_{X1} = 0.25$	$x \in [2,3]$ $p_{X2} = 0.5$	$x \in [3,4]$ $p_{X3} = 0.25$	$\leftrightarrow \quad \updownarrow$ $X \quad Y$

Thus, we can see that the joint distribution is affected by the dependency relationship between X and Y. If we don't know the relationship between X and Y, we can't determine the joint distribution in this matrix. But we can infer some things about the result variable from this matrix. For example, consider  $z=5$ . It only occurs in the following grey cells.

**Table 1.4. Joint distribution for specified value.**

$z \in [3,5]$ $p_{11} = ?$	$z \in [4,6]$ $p_{12} = ?$	$z \in [5,7]$ $p_{13} = ?$	$y \in [2,3]$ $p_{Y1} = 0.5$
$z \in [4,6]$ $p_{21} = ?$	$z \in [5,7]$ $p_{22} = ?$	$z \in [6,8]$ $p_{23} = ?$	$y \in [3,4]$ $p_{Y2} = 0.3$
$z \in [5,7]$ $p_{31} = ?$	$z \in [6,8]$ $p_{32} = ?$	$z \in [7,9]$ $p_{33} = ?$	$y \in [4,5]$ $p_{Y3} = 0.2$
$x \in [1,2]$ $p_{X1} = 0.25$	$x \in [2,3]$ $p_{X2} = 0.5$	$x \in [3,4]$ $p_{X3} = 0.25$	$\leftrightarrow \quad \updownarrow$ $X \quad Y$



As previous stated, we don't know the exact probability for  $z \leq 5$ . But we can think about what are the possible probabilities for  $z \leq 5$ . As this matrix shows, only grey cells contribute to the probability of  $z \leq 5$ . We would like to determine the maximum probability and the minimum probability. To get the maximum value, all cells in which  $Z$  can be  $\leq 5$  will have their probabilities summed. To obtain the minimum value, only cells, in which  $Z$  must be  $\leq 5$ , will have their probabilities summed. For example, considering cell  $p_{12}$  for  $p\{Z \leq 5\}$ , when we calculate the maximum value, this cell must be counted because  $Z$  can be  $\leq 5$  in this cell. But for the minimum value, we don't count this cell because  $Z$  might not  $\leq 5$  in this cell. This way, we can find the possible range of cumulative probabilities for various values of  $Z$ . We can find the maximum possibility and minimum possibility for every value of  $Z$  and connect all these points to get 2 curves: a top curve and a bottom curve. All the CDFs that are possible for  $Z$ , must belong between these two curves.

In this example,  $Z$ 's range is from 3 to 9. It is clear that the probability for  $Z < 3$  is zero and for  $Z > 9$  is 1. The following part discusses the probability of  $Z \leq 4$ .

Maximum: We try to find all the cells in which this situation may occur. From the previous table, these cells are  $p_{11}$ ,  $p_{12}$ , and  $p_{21}$ . So the maximum value should be the maximum value for the sum of  $p_{11}$ ,  $p_{12}$ , and  $p_{21}$ .

Minimum: To obtain the minimum, we will find all the cells in which  $Z$  must be  $\leq 4$ . In this table, there are none. Although  $p_{11}$ ,  $p_{12}$ , and  $p_{21}$  may satisfy  $Z \leq 4$ , they also might not. For example, the whole probability for the cell might be concentrated at the high bound of its range. So there is no cell in which  $Z$  must be  $\leq 4$ .

Summarizing the above analysis, we can define a way to tell which cells contribute to the maximum and minimum probability values.

Maximum: all the cells in which the low bound is not greater than the value of  $Z$  contribute to the max value.

Minimum: all the cells in which the high bound is not greater than the value of  $Z$  must contribute to the min value.

After finding all the cells satisfying the max (or min) condition, we will calculate the sum of the probabilities of these cells. Based on the previous table, there exist constraints for the probabilities  $P_{ij}$ . It is clear that the sum of the  $P_{ij}$ 's in a row or column can't go over the

marginal probability of that row or column. These constraints can be described as follows:

Row Constraints:  $\sum_{j=1}^3 p_{ij} = p_{Yi}$  for  $i=1$  to 3

Column Constraints:  $\sum_{i=1}^3 p_{ij} = p_{Xj}$  for  $j=1$  to 3

Therefore, the question becomes: find the maximum and minimum value for the sum of cells under these constraints. For the case  $Z \leq 4$ , we can describe these questions using mathematically:

Maximum - make the sum of the specified cells' value big enough, that is, find

$$\max (p_{11} + p_{12} + p_{21})$$

such that:

$$\sum_{j=1}^3 p_{ij} = p_{Yi} \text{ for } i=1 \text{ to } 3$$

$$\text{and } \sum_{i=1}^3 p_{ij} = p_{Xj} \text{ for } j=1 \text{ to } 3.$$

Minimum - make the sum of specified cells' value small enough, that is, find

$$\min \left( \sum_{i=1}^3 \sum_{j=1}^3 0 * p_{ij} \right)$$

such that:

$$\sum_{j=1}^3 p_{ij} = p_{Yi} \text{ for } i=1 \text{ to } 3$$

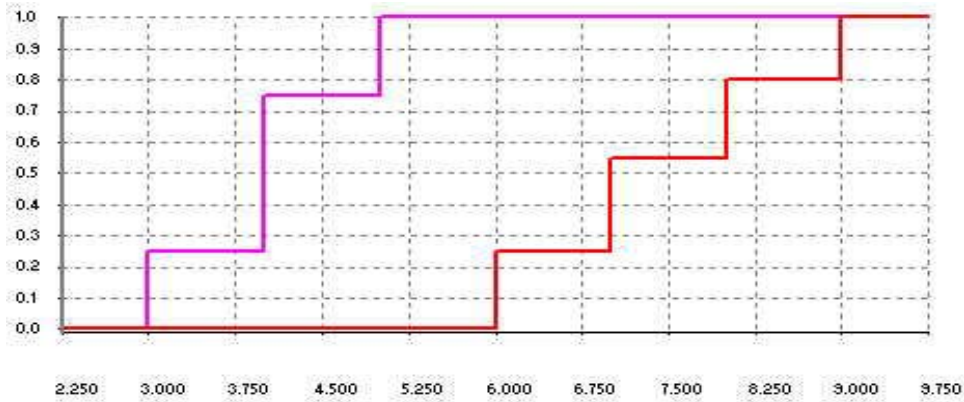
$$\text{and } \sum_{i=1}^3 p_{ij} = p_{Xj} \text{ for } j=1 \text{ to } 3$$

For these two optimization questions, linear programming is the best tool to find the solution. This way, we can find the probability range for the specified value of Z. The following table shows the probabilities for various values of Z.

**Table 1.5. Probabilities for result variable.**

Z range	Maximum probability	Minimum probability
$Z < 3$	0	0
$Z \leq 3$	0.25	0
$Z \leq 4$	0.75	0
$Z \leq 5$	1	0
$Z \leq 6$	1	0.25
$Z \leq 7$	1	0.55
$Z \leq 8$	1	0.8
$Z \leq 9$	1	1
$Z \leq 10$	1	1

From this table, we can draw two curves, a top curve and a bottom curve, using the maximum and minimum probabilities shown Z value. These two curves also can be called envelopes for the CDF of derived variable Z because the CDF for derived variable Z must be between these 2 curves whatever the relationship between X and Y is. This figure shows the final result.

**Figure 1.1. Probability bounds**

## 2 Narrowing the envelopes around results using correlation

In the previous chapter we noted an important factor: correlation. If one knows something about correlation, it would be good to be able to use it. We describe how next.

### 2.1 *Facts about correlation*

Correlation is used to measure the degree of correspondence between random variables. To describe this kind of relationship, there are a number of methods. For example, we can consider the linear relationship between two random variables, or the square relationship. Currently, the most popular correlation coefficient is called Pearson correlation or product-moment correlation. It is used to measure the strength of the linear relationship between two random variables. It is defined as

$$\rho = \frac{E[(X - EX)(Y - EY)]}{\sqrt{D(X)D(Y)}}$$

Here  $D(X)$  is  $X$ 's variance and  $D(Y)$  is  $Y$ 's variance.  $E$  means expectation.

It is clear that  $-1 \leq \rho \leq 1$ . Correlations can be classified into 3 types: positive correlation ( $\rho > 0$ ), meaning there is a direct linear correlation between the R.V.'s), negative correlation ( $\rho < 0$ ), meaning there is an inverse linear correlation between the R.V.'s), and no correlation ( $\rho = 0$ ), meaning there is no apparent linear correlation between the R.V.'s). There also are 2 special cases: perfect positive correlation ( $\rho = 1$ ) and perfect negative correlation ( $\rho = -1$ ). For perfect positive correlation, we can get:

- $P[Y=aX+b]=1$ , for some  $b$  and some  $a > 0$ .
- When  $X$  takes on its largest value,  $Y$  also does.

For perfect negative correlation, we can get:

- $P[Y=aX+b]=1$ , for some  $b$  and some  $a < 0$ .
- When  $X$  takes on its largest value,  $Y$  has its smallest value.

## 2.2 Joint distributions

A joint distribution is used to describe the detailed dependency between two R.V.'s. From the joint distribution, we can get the correlation. But correlation doesn't imply a specific joint distribution, so we can't get the joint distribution from a value of correlation, in general.

## 2.3 Interval-valued correlations

When the correlation is unknown we use linear programming to find CDF envelopes. If we know the correlation for two operands, we would like to use it to determine additional constraints for the linear programming problem. In another words, we wish to decrease the feasible solution space and get a better solution.

According to the definition of correlation, for two random variables  $x$  and  $y$ , the correlation is

$$\rho = \frac{E[(x - Ex)(y - Ey)]}{D(x)D(y)} = \frac{E[(x - Ex)(y - Ey)]}{\sqrt{E[(x - Ex)^2]} \sqrt{E[(y - Ey)^2]}}$$

Where  $Ex$  and  $Ey$  are the means for variable  $x$  and  $y$ .

Using the following formulas, we can reduce (1):

$$\begin{aligned} E[(x - Ex)(y - Ey)] &= E[xy - yEx - xEy + Ex * Ey] \\ &= Exy - Ey * Ex - Ex * Ey + Ex * Ey = Exy - Ex * Ey \end{aligned}$$

Also,

$$\begin{aligned} E[(x - Ex)^2] &= E[x^2 - 2xEx + Ex * Ex] = Ex^2 - 2Ex * Ex + Ex * Ex \\ &= Ex^2 - (Ex)^2 \end{aligned}$$

So previous formula becomes

$$\rho = \frac{E[(x - Ex)(y - Ey)]}{\sqrt{E[(x - Ex)^2]} \sqrt{E[(y - Ey)^2]}} = \frac{Exy - Ex * Ey}{\sqrt{(Ex^2 - (Ex)^2)(Ey^2 - (Ey)^2)}}$$

Using the definition of mean, when variable  $x$  is discrete,

$$Ex = \sum_i x_i * p_i \text{ where } p_i(x = x_i) = p_i$$

When variable  $x$  is continuous, and has density function  $f$ , then

$$Ex = \int x f(x) dx.$$

In the DEnv algorithm, we don't care if a random variable is discrete or continuous. We use bars to discretize the distribution. This method has the following characteristics:

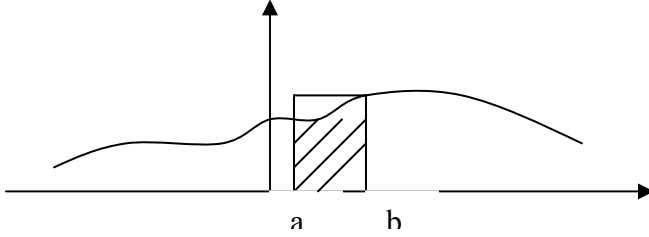
1. Bars may overlap.
2. Histograms are a special case of collections of bars.
3. A bar describes the probability of an interval containing the value of a variable.
4. No assumption is made about the distribution over the interval of bar.

We now extend the definition of mean to intervals. We can handle it like the discrete case. So

$$EX = \sum_i X_i * P_i \text{ where } P_i(x \in X_i) = P_i.$$

It's clear that the mean of variable x must be in EX.

When x is continuous, we also can get the mean based on the following argument.



Consider some bar in the discretization of variable x whose distribution function is  $f(x)$ . The probability that it is in  $[a, b]$  is the area of  $f(x)$  between a and b.

$$P(a \leq x \leq b) = \int_a^b f(x) dx$$

We can partition the domain of variable x into many intervals such as this one, denoting them  $X_i$ . These intervals do not overlap. They together will cover the range for variable x. So the mean of variable x becomes

$$Ex = \int xf(x) dx = \sum_i \int_{X_i} xf(x) dx$$

Consider one item in the previous formula, assuming  $X_i$  is  $[a, b]$  as in the previous figure.

$$\int_{X_i} xf(x) dx = \int_{[a, b]} xf(x) dx \geq \int_{[a, b]} af(x) dx = a * \int_{[a, b]} f(x) dx = a * p_i.$$

Similarly, we also get

$$\int_{X_i} xf(x)dx = \int_{[a,b]} xf(x)dx \leq \int_{[a,b]} bf(x)dx = b * \int_{[a,b]} f(x)dx = b * p_i .$$

So,  $\int_{X_i} xf(x)dx$  must belong to  $X_i * p_i$ . So,  $Ex$  contains the mean of interval variable  $X$ .

If the intervals overlap, the width of the mean is wider than in the non-overlapped case. So the mean of the non-overlapped intervals is a subset of that of the overlapped intervals. Thus the  $Ex$  belongs to mean of interval variable  $X$  in this case too.

Here is how we use this result. If any intervals overlap, it means at least two intervals overlap. We know the left endpoint for one interval is not bigger than that of the non-overlapped condition and the right endpoint for one interval is not less than that of the non-overlapped condition.

## 2.4 Legal and illegal correlation values

In the current software, the user can input any value of correlation from  $-1$  to  $1$ . But in fact, for some marginal distributions, there are correlation values which will not be exhibited by any joint distribution. In fact, the constraints coming from setting correlation to an impossible value should be conflict with the constraints coming from the marginals of the joint distribution matrix.

From the definition of correlation, we can get this formula for  $Exy$ :

$Exy = Ex * Ey + \rho \sqrt{(Ex^2 - (Ex)^2)(Ey^2 - (Ey)^2)}$  Let  $f(x,y)=Exy$ , so  $f(x,y)$  is a real function of  $x$  and  $y$ . We can rewrite  $Exy$  with intervals  $X$  and  $Y$ .

$$\begin{aligned} f(X,Y) &= EX * EY + \rho \sqrt{(EX^2 - (EX)^2)(EY^2 - (EY)^2)} \\ &= \sum_i X_i P_{xi} \sum_j Y_j P_{yj} + \rho \sqrt{(\sum_i X_i^2 P_{xi} - (\sum_i X_i P_{xi})^2)(\sum_j Y_j^2 P_{yj} - (\sum_j Y_j P_{yj})^2)} . \end{aligned}$$

The corresponding real function is

$$f(x,y) = \sum_i x_i p_{xi} \sum_j y_j p_{yj} + \rho \sqrt{(\sum_i x_i^2 p_{xi} - (\sum_i x_i p_{xi})^2)(\sum_j y_j^2 p_{yj} - (\sum_j y_j p_{yj})^2)}$$

where  $x_i \in X_i$  and  $y_j \in Y_j$ . If  $\rho$  is an interval, it becomes another variable for function  $f$ .

### 2.4.1 Solution

The software should provide a way to help the user to set a reasonable correlation. To do this, first, the software must figure out the range of possible correlations for the current

random variables. Then the software can display this information. It then only accepts values intersecting with this range.

As mentioned before, there are 2 kinds of constraints, one coming from the marginals of the joint distributions matrix and another coming from the correlation setting. The joint distribution matrix marginals are assumed correct. So, the constraints coming from it are a given. If constraints coming from a correlation setting conflict with them, they must be in error. Constraints coming from the matrix are primary and constraints coming from correlation should be considered secondary.

Consider a joint distribution matrix for an operation  $\otimes$ .

**Table 2.1. Joint distribution matrix.**

	Y1	...	Ym	
X1	$p_{11}$	...	$p_{1m}$	$p_{x1}$
...	...	...	...	...
Xn	$p_{n1}$		$p_{nm}$	$p_{xn}$
	$p_{y1}$	...	$p_{ym}$	

We can get  $E_{xy}$  as follows:

$E_{xy} = \sum_{i=1}^n \sum_{j=1}^m X_i Y_j p_{ij}$  where  $X_i$  and  $Y_j$  are interval values.  $p_{ij}$  is the probability assigned to cell  $ij$ . We use underlining to indicate the low bound of an interval and overlining to indicate the high bound of an interval. We can get the bounds of  $E_{xy}$  as follows:

$$\underline{E_{xy}} = \underline{x_1 y_1} p_{11} + \underline{x_1 y_2} p_{12} + \underline{x_1 y_3} p_{13} + \dots + \underline{x_2 y_1} p_{21} + \underline{x_2 y_2} p_{22} + \underline{x_2 y_3} p_{23} + \dots + \underline{x_n y_m} p_{nm}$$

$$\overline{E_{xy}} = \overline{x_1 y_1} p_{11} + \overline{x_1 y_2} p_{12} + \overline{x_1 y_3} p_{13} + \dots + \overline{x_2 y_1} p_{21} + \overline{x_2 y_2} p_{22} + \overline{x_2 y_3} p_{23} + \dots + \overline{x_n y_m} p_{nm}$$

From this, we get two linear programming problems:

Min  $\underline{E_{xy}} = \underline{x_1 y_1} p_{11} + \underline{x_1 y_2} p_{12} + \underline{x_1 y_3} p_{13} + \dots + \underline{x_2 y_1} p_{21} + \underline{x_2 y_2} p_{22} + \underline{x_2 y_3} p_{23} + \dots + \underline{x_n y_m} p_{nm}$   
subject to:

$$\text{row Constraints: } \sum_{j=1}^m p_{ij} = p_{xi} \text{ for } i=1 \text{ to } n;$$

$$\text{column Constraints: } \sum_{i=1}^n p_{ij} = p_{yj} \text{ for } j=1 \text{ to } m.$$



Max  $\overline{Exy} = \overline{x_1 y_1 p_{11}} + \overline{x_1 y_2 p_{12}} + \overline{x_1 y_3 p_{13}} + \dots + \overline{x_2 y_1 p_{21}} + \overline{x_2 y_2 p_{22}} + \overline{x_2 y_3 p_{23}} + \dots + \overline{x_n y_m p_{nm}}$   
 subject to:

row Constraints:  $\sum_{j=1}^m p_{ij} = p_{xi}$  for  $i=1$  to  $n$ ;

column Constraints:  $\sum_{i=1}^n p_{ij} = p_{yj}$  for  $j=1$  to  $m$ .

Solving these two linear programming problems, we can get the bounds of  $\overline{Exy}$ , call these numbers  $\underline{k}$  and  $\overline{k}$ . We also know

$$\overline{Exy} = \sum_i x_i p_{xi} \sum_j y_j p_{yj} + \rho \sqrt{(\sum_i x_i^2 p_{xi} - (\sum_i x_i p_{xi})^2)(\sum_j y_j^2 p_{yj} - (\sum_j y_j p_{yj})^2)}$$

where  $x_i \in X_i$  and  $y_j \in Y_j$ .

In this formula, only  $\rho$  is an unknown range. Now the problem becomes solving for  $\overline{Exy}$ . The minimum should be the minimum value of  $\rho$ . The maximum should be the maximum value of  $\rho$ . So the problem is transformed into finding the root range of a nonlinear function.

## 2.4.2 Approximate solution

From  $f(x,y) = \overline{x} \cdot \overline{y} + \rho \sqrt{(\overline{x^2} - \overline{x}^2)(\overline{y^2} - \overline{y}^2)}$ , in most cases,  $\overline{x} \cdot \overline{y}$  is greater than  $\sqrt{(\overline{x^2} - \overline{x}^2)(\overline{y^2} - \overline{y}^2)}$ . So we just consider  $\overline{x} \cdot \overline{y}$ . It is obvious that it is an increasing function of  $x$  and  $y$ . Assigning the minimum values to  $x$  and  $y$ , and the maximum value possible for  $f(x,y)$ , we can obtain the maximum value of  $\rho$ . Assigning the maximum values to  $x$  and  $y$ , and the minimum value to  $f(x,y)$ , we can get minimum value of  $\rho$ .

## 2.5 Additional constraints gotten from correlation

When the user sets the correlation range, we know the range of every variable in formula  $f(x,y)$ . Under this situation, we can get the range of  $f(x,y)$ . This range of  $f(x,y)$  is thus controlled by the user. At the same time, we know another range for  $f(x,y)$  which is derived from the joint distribution matrix. As previous noted, the range derived from the joint distribution matrix is considered given. So it is always correct. The range coming from the

user must be intersected with this range. From this restriction, we can get additional constraints for linear programming.

Obviously, formula  $f(x,y)$  is non-linear, so we use non-linear optimization to do minimization and maximization on it. Using a penalty function transforms a constrained optimization problem to a non-constrained problem. We also can get the first and second derivative for this function. Call the values obtained  $f_{min}$  and  $f_{max}$ . So we get  $f(x,y)=[f_{min},f_{max}]$ .

From the previous section, we know another range for  $f(x,y)$ ,  $[\underline{k}, \overline{k}]$ , from the joint distribution matrix. It is obvious that these two ranges must intersect; otherwise, the user input is not possible. These two ranges both are intervals. If the following conditions are satisfied, these two intervals must be intersected:

$f_{max} \geq \underline{k}$  and  $f_{min} \leq \overline{k}$ . Since

$$\underline{k} = \underline{x_1 y_1} p_{11} + \underline{x_1 y_2} p_{12} + \underline{x_1 y_3} p_{13} + \dots + \underline{x_2 y_1} p_{21} + \underline{x_2 y_2} p_{22} + \underline{x_2 y_3} p_{23} + \dots + \underline{x_n y_m} p_{nm}$$

$$\overline{k} = \overline{x_1 y_1} p_{11} + \overline{x_1 y_2} p_{12} + \overline{x_1 y_3} p_{13} + \dots + \overline{x_2 y_1} p_{21} + \overline{x_2 y_2} p_{22} + \overline{x_2 y_3} p_{23} + \dots + \overline{x_n y_m} p_{nm}$$

we know  $f_{min}$  and  $f_{max}$ . So we get an additional two linear constraints for the linear programming problems based on correlation:

$$\underline{x_1 y_1} p_{11} + \underline{x_1 y_2} p_{12} + \underline{x_1 y_3} p_{13} + \dots + \underline{x_2 y_1} p_{21} + \underline{x_2 y_2} p_{22} + \underline{x_2 y_3} p_{23} + \dots + \underline{x_n y_m} p_{nm} \leq f_{max}$$

$$\overline{x_1 y_1} p_{11} + \overline{x_1 y_2} p_{12} + \overline{x_1 y_3} p_{13} + \dots + \overline{x_2 y_1} p_{21} + \overline{x_2 y_2} p_{22} + \overline{x_2 y_3} p_{23} + \dots + \overline{x_n y_m} p_{nm} \geq f_{min}$$

## 2.6 Nonlinear optimization to remove excess width

From the previous section, we saw that  $f(X,Y)$  is an interval, not a real number. In interval mathematics, it is called an interval function (Ramon E. Moore, 1966). In evaluating an interval function, excess width may happen. Different function formats will result the different values for function although they are the same function in the real domain.

From the term  $\rho * \sqrt{D(X) * D(Y)} + E(X)E(Y)$ , we defined the corresponding function  $f(x,y)$ :  $f(x,y) = Ex * Ey + \rho \sqrt{(Ex^2 - (Ex)^2)(Ey^2 - (Ey)^2)}$ . First, we can consider  $f(x,y)$  as a real function of variables  $x$  and  $y$ . If we replace  $x$  and  $y$  with intervals  $X$  and  $Y$ , it becomes an interval function.

Based on the rule “cancellation or reduction of the number of occurrences of a variable before interval evaluation”, if the number of occurrences of each variable is only one, evaluating an interval function cannot result in excess width. However it is impossible to use this rule for this function. Instead, we can avoid this problem by evaluating this function in the real domain using real numbers  $x$  belonging to interval  $X$ . So we can use the minimum value and the maximum value of this real function as the way to get bounds on the interval.

Now we rewrite the formula with intervals  $X$  and  $Y$ .

$$\begin{aligned} f(X, Y) &= EX * EY + \rho \sqrt{(EX^2 - (EX)^2)(EY^2 - (EY)^2)} \\ &= \sum_i X_i P_{xi} \sum_j Y_j P_{yj} + \rho \sqrt{(\sum_i X_i^2 P_{xi} - (\sum_i X_i P_{xi})^2)(\sum_j Y_j^2 P_{yj} - (\sum_j Y_j P_{yj})^2)} \end{aligned}$$

The corresponding real function is

$$f(x, y) = \sum_i x_i p_{xi} \sum_j y_j p_{yj} + \rho \sqrt{(\sum_i x_i^2 p_{xi} - (\sum_i x_i p_{xi})^2)(\sum_j y_j^2 p_{yj} - (\sum_j y_j p_{yj})^2)}$$

Here  $x_i \in X_i$  and  $y_j \in Y_j$ . If  $\rho$  is an interval number, it becomes another variable for function  $f$ .

Obviously,  $f(x, y)$  is a non-linear function. We use non-linear optimization to figure out the minimum and maximum. But this optimization question is restricted to a special region, the intervals for the  $x$ 's and  $y$ 's.

## 2.7 Improving results by adding constraints to LP

Based on the above discussion, we get another two constraints for LP after calculating the interval  $k$ . From the joint distribution matrix,

**Table 2.2. Joint distribution for  $X$  and  $Y$ .**

	[...]	...	[...]	$X$
[...]	$p_{11}$	...	$p_{1n}$	$px_1$
...	...	...	...	...
[...]	$p_{m1}$	...	$p_{mn}$	$px_m$
$Y$	$py_1$	....	$py_n$	1

we get the LP model:

$$\text{Minimize } Z = \sum_{i,j \in \Omega} p_{ij}$$

$$\text{subject to: } \begin{cases} \sum_j p_{ij} = px_i, i = 1 \dots m \\ \sum_i p_{ij} = py_j, j = 1 \dots n \\ p_{ij} \geq 0, px_i \geq 0, py_j \geq 0, \\ \sum px_i = 1, \sum py_j = 1 \end{cases}$$

To these we add the two constraints implied by the correlation. When the two constraints,  $\sum \overline{a_{ij}} p_{ij} = \overline{k}$  and  $\sum \underline{a_{ij}} p_{ij} = \underline{k}$ , are added to the LP, the transportation simplex method can't handle this augmented model because we can't put these two constraints into the balanced transportation tableau.

So we use the simplex method to solve the problem. The speed of calculation is very important. This is discussed later.

## 2.8 Improved simplex method

Consider the standard LP question:

$$\text{Min } Z = CX$$

Subject to:  $AX = b$ ,  $x_i \geq 0$  and  $b_i \geq 0$  for  $i=1$  to  $n$ .

Here  $C = (c_1, \dots, c_n)$  is a row vector,  $X = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}$  is a column vector.  $A = (P_1, \dots, P_n)$

and  $P_i = \begin{pmatrix} a_{1i} \\ \dots \\ a_{mi} \end{pmatrix}$ . So, A is an  $m \times n$  matrix and  $b = \begin{pmatrix} b_1 \\ \dots \\ b_m \end{pmatrix}$  is a column vector.

We can transform the maximization problem to a minimization problem through the following approach.

$$\text{Max } Z = CX \Leftrightarrow \text{Min } Y = -Z = -CX$$

The constraints are unchanged.

Based on the simplex method,  $A$  is split into  $(A_B \ A_N)$ .  $A_B$  has the coefficients for the basic variables (assuming there are  $m$  basic variable from  $x_1$  to  $x_m$ ), and  $A_N$  has the coefficients for the non-basic variables (from  $x_{m+1}$  to  $x_n$ ).  $X$  is also separated into  $\begin{pmatrix} X_B \\ X_N \end{pmatrix}$ .

$$\text{So } AX=b \text{ becomes } (A_B \ A_N) \begin{pmatrix} X_B \\ X_N \end{pmatrix} = b.$$

$$\Leftrightarrow A_B * X_B + A_N * X_N = b$$

$$\Leftrightarrow X_B = A_B^{-1} * (b - A_N * X_N)$$

Here  $A_B^{-1}$  means the inverse matrix of  $A_B$ . In other word,  $A_B * A_B^{-1} = I$  where  $I$  is the unit

matrix. For example,  $\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$  is a 3\*3 unit matrix.

So, the objective function becomes

$$\begin{aligned} Z &= C * X = (C_B \ C_N) * \begin{pmatrix} X_B \\ X_N \end{pmatrix} \\ &= C_B * X_B + C_N * X_N \\ &= C_B * A_B^{-1} (b - A_N * X_N) + C_N * X_N \\ &= C_B A_B^{-1} b + (C_N - C_B A_B^{-1} A_N) X_N \end{aligned}$$

Let us see an example.

Minimize  $z = 3x_1 - x_2 - 7x_3 + 3x_4 + x_5$

$$\text{Subject to: } \begin{cases} 5x_1 - 4x_2 + 13x_3 - 2x_4 + x_5 = 20 \\ x_1 - x_2 + 5x_3 - x_4 + x_5 = 8 \\ x_i \geq 0, i = 1 \dots 5 \end{cases}$$

$$\text{Here } C = (3 \ -1 \ -7 \ 3 \ 1), \ X = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}, \ A = \begin{vmatrix} 5 & -4 & 13 & -2 & 1 \\ 1 & -1 & 5 & -1 & 1 \end{vmatrix} \text{ and } b = \begin{pmatrix} 20 \\ 8 \end{pmatrix}.$$

If we assume  $x_1$  and  $x_2$  are the basic variables, we get  $X_B = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, X_N = \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix},$

$C_B = (3 \ -1), C_N = (-7 \ 3 \ 1), A_B = \begin{bmatrix} 5 & -4 \\ 1 & -1 \end{bmatrix}, A_N = \begin{bmatrix} 13 & -2 & 1 \\ 5 & -1 & 1 \end{bmatrix}.$  We also get

$$A_B^{-1} = \begin{bmatrix} 1/3 & -2/3 \\ 1/6 & -5/6 \end{bmatrix}, X_B = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = A_B^{-1}(b - A_N X_N).$$

The following discussion will be based on the previous definition and equations, and also in part on Qian and Murty (1985).

### 2.8.1 How to find the initial feasible solution

For the standard LP question, if you can find a unit  $m \times m$  matrix in A, you let this matrix be  $A_B$  by multiplying one row by a constant and adding it to another row, repeating as needed. Set  $X_N$  (non-basic variables) to zero (that is  $x_{m+1}, x_{m+2}, \dots, x_n$  all equal 0). Then

$$X_B = A_B^{-1}(b - A_N X_N) = A_B^{-1} * b = I * b = b \text{ because since A is a unit matrix, so is } A^{-1}.$$

Then,  $x_i = b_i \geq 0, (i = 1 \dots m)$  is a feasible solution although it is probably not the optimal solution.

If you can't find a unit matrix, you can choose a sub-matrix ( $m \times m$ ) of A which is nonsingular (meaning that the determinant of the matrix doesn't equal zero and the rank of the matrix is m), and every  $x_i$  of  $X_B = A_B^{-1} * b$  is not less than 0. Under this condition, it is a feasible solution.

But frequently, it is not so easy. Therefore artificial variables are introduced.

To  $AX = b$ , we add the artificial variables  $Y = \begin{pmatrix} y_1 \\ \dots \\ y_m \end{pmatrix} \geq 0$ , and revise the equation to

$AX + IY = b$ . In the objective function, the coefficients of Y should be very large positive real numbers. Through this way, the minimizing objective function will be unaffected by artificial variables Y. Still we can use Y as the initial feasible solution. Importing the artificial variables just provides an easy way to get an initial feasible solution.

### 2.8.2 How to decide the termination condition and entering variable

Now consider optimization of  $Z$ . Let  $W = C_N - C_B A_B^{-1} A_N = (w_1, \dots, w_{n-m})$ . Here  $w_i$  is the coefficient of  $x_{m+i}$  and describes the coefficient of a non-basic variable in the objective function. If we want to make

$Z = C * X = C_B A_B^{-1} b + (C_N - C_B A_B^{-1} A_N) X_N = C_B A_B^{-1} b + W X_N$  smaller, we must hope to find the negative elements of  $W$  because all elements of  $X_N$  are positive. From this

discussion, we can derive the termination rule for an iterative optimization process.

1. If every element  $w_i$  of  $W$  is not less than 0, then the current solution is optimal.
2. If at least one element of  $W$  is negative, we continue to search for the optimal solution. Let  $w_k = \min(w_i \mid w_i < 0)$ . This means if every non-basic variable changes by the same factor, value  $w_k * x_{m+k}$  will have the maximum effect in minimizing the value of  $Z$ . So let non-basic variable  $x_{m+k}$  be the entering variable (entering the basic variable set from non-basic variable set).
3. If  $A_B^{-1} P_{m+k} \leq 0$ , there is no solution ( $k$  is the entering variable index, and  $P_{m+k}$ , belonging to  $A_N = (P_{m+1}, \dots, P_n)$ , is the coefficient for non-basic variable  $x_{m+k}$ ).

**Proof:**

From  $X_B = A_B^{-1} * (b - A_N * X_N)$ , assuming the entering variable  $x_{m+k}$  does not equal 0 and other non-basic variables still equal 0, let  $x_{m+k}$  equal  $\alpha$  and be greater than 0. Then

$$\begin{aligned}
 X_B &= A_B^{-1} * b - A_B^{-1} A_N * X_N \\
 &= A_B^{-1} * b - A_B^{-1} (P_{m+1}, \dots, P_n) \begin{pmatrix} x_{m+1} \\ \dots \\ x_n \end{pmatrix} \\
 &= A_B^{-1} * b - A_B^{-1} P_{m+k} x_{m+k} \\
 &= A_B^{-1} * b - \alpha A_B^{-1} P_{m+k}
 \end{aligned}$$

Because  $A_B^{-1} P_{m+k} \leq 0$ ,  $X_B$  still are greater than 0, and  $X_N = 0$  except for  $x_{m+k} = \alpha$ .

So it is a feasible solution. Consider the objective function:

$$\begin{aligned}
 Z &= C_b A_B^{-1} b + (C_n - C_b A_B^{-1} A_N) X_N \\
 &= C_b A_B^{-1} b + (w_1, \dots, w_{n-m}) \begin{pmatrix} x_{m+1} \\ \dots \\ x_n \end{pmatrix} \\
 &= C_b A_B^{-1} b + w_k * \alpha
 \end{aligned}$$

Because  $w_k$  is less than 0, if  $\alpha \rightarrow +\infty$ ,  $Z \rightarrow -\infty$ .

So, there is no minimum value for the objective function.

Let us see an example:

Minimize  $z = -x_1 - x_2$

$$\text{Subject to: } \begin{cases} -2x_1 + x_2 + x_3 = 4 \\ x_1 - x_2 + x_4 = 2 \\ x_i \geq 0, i = 1 \dots 4 \end{cases}$$

We choose  $x_2$  and  $x_4$  as basic variables. Then  $A_B = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$ ,  $A_N = \begin{bmatrix} -2 & 1 \\ 1 & 0 \end{bmatrix}$ ,  $C_B = (-1 \ 0)$

and  $C_N = (-1 \ 0)$ .  $A_B^{-1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ , so  $W = C_N - C_B A_B^{-1} A_N = (-3 \ 1)$ . We can choose  $x_1$  as

the entering variable. We get  $A_B^{-1} P_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{pmatrix} -2 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ -1 \end{pmatrix} < 0$ . Now let  $x_1$  equal  $\beta > 0$ . So

$$X_B = A_B^{-1} b - A_B^{-1} A_N * X_N = \begin{pmatrix} 4 \\ 6 \end{pmatrix} - \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix} - x_1 \begin{pmatrix} -2 \\ -1 \end{pmatrix} = \begin{pmatrix} 4 + 2x_1 \\ 6 + x_1 \end{pmatrix}$$

Here  $X_B$  is a feasible solution if  $x_1 \geq 0$ . But

$Z = (-1 \ -1) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = -x_1 - (4 + 2x_1) = -4 - 3x_1$ . If  $x_1 = \beta \rightarrow \infty$ , then  $Z \rightarrow -\infty$ . So there is no minimum value for  $Z$ .

Based on the previous discussion, there are three conditions that can occur during the iterative procedure.

1. Finding the solution
2. Continuing to try minimizing  $Z$
3. No minimization solution

### 2.8.3 How to determine the leaving variable

Let  $X_B$  be a feasible solution. So  $A_B * X_B = b$ . Here  $A_B = (P_1 \ \dots \ P_m)$ . We know  $A_B$  is nonsingular, so  $P_1$  to  $P_m$  are the independent vectors. The other vectors  $P_{m+1}$  to  $P_n$  are linearly dependent on  $P_1$  to  $P_m$ . Therefore we can get

$$P_{m+j} = \sum_{i=1}^m \alpha_{i,m+j} * P_i$$



$$\Rightarrow P_{m+j} - (P_1, \dots, P_m) * \begin{pmatrix} \alpha_{1,m+j} \\ \dots \\ \alpha_{m,m+j} \end{pmatrix} = 0$$

From  $A_B * X_B = b$ , we get

$$(P_1 \dots P_m) * X_B = b.$$

Let  $\beta$  a positive real number. Then

$$(P_1, \dots, P_m) * X_B + \beta(P_{m+j} - (P_1, \dots, P_m) * (\alpha_{1,m+j}, \dots, \alpha_{m,m+j})') = 0$$

$$\Rightarrow (P_1, \dots, P_m) * (X_B - \beta(\alpha_{1,m+j}, \dots, \alpha_{m,m+j})') + \beta P_{m+j} = 0.$$

Let  $x_{m+j}$  replace a variable in  $X_B$ . We can get a new feasible solution if we set suitable values for  $X$  and make sure  $x_i \geq 0$ . We can get a suitable solution from the previous formulation through setting the new  $X_B$  to equal  $X_B - \beta(\alpha_{1,m+j}, \dots, \alpha_{m,m+j})'$ . We will let one element that equals 0 to be replaced by  $x_{m+j}$ . To assure the other variables in  $X_B$  stay positive, we can choose a suitable  $\beta$ . Let

$$\beta = \min\left(\frac{x_i}{\alpha_{i,m+j}} \mid \alpha_{i,m+j} > 0\right) = \frac{x_l}{\alpha_{l,m+j}}.$$

This implies that  $x_l$  is the leaving variable and entering variable  $x_{m+j} = \frac{x_l}{\alpha_{l,m+j}}$ .

Now we can apply this result. Based on

$$X_B = A_B^{-1} * (b - A_N * X_N)$$

$$= A_B^{-1} b - A_B^{-1} A_N * X_N$$

$$= A_B^{-1} b - x_{m+j} * A_B^{-1} P_{m+j}$$

we know  $x_{m+j}$  is the entering variable. We can determine the leaving variable by choosing the minimum  $\beta$  using the equation

$$\beta = \min\left(\frac{(A_B^{-1} b)_i}{(A_B^{-1} P_{m+j})_i} \mid (A_B^{-1} P_{m+j})_i > 0\right) = \frac{(A_B^{-1} b)_l}{(A_B^{-1} P_{m+j})_l}.$$

This implies that the leaving variable is  $x_l$ .

## 2.8.4 Decreasing computing

The simplex method is a good way to solve linear programming. But it can have computational complexity problems.

From the previous discussion, we can see the main complexity problem focuses on the inverse matrix  $A_B$ . If we can find a better way to compute it, we can get better efficiency. A simple approach is to find the relationship between the two  $A_B$  in the closing steps. If we can use the previous  $A_B$  to speed computing the next  $A_B$ , it will help. If the original  $A_B = (P_1 \dots P_m)$ , the new  $A_B$  is  $A_B = (P_1 \dots P_{l-1} P_{m+k} P_{l+1} \dots P_m)$ . There is only one different column. So the coefficient of the leaving variable is replaced with that of the entering variable. We can guess there is a relationship between these two  $A_B$ . From  $X_B^{old} = A_B^{old^{-1}} b$ ,  $X_B^{new} = A_B^{old^{-1}} b - x_{m+k} * A_B^{old^{-1}} P_{m+k} = A_B^{new^{-1}} b$ , and basic variable  $x_l$  is replaced with  $x_{m+k}$ ,

$$A_B^{new} = A_B^{old} C.$$

Then

$$A_B^{new^{-1}} = C^{-1} A_B^{old^{-1}} = D A_B^{old^{-1}}$$

So if we can find D, the inverse of C, we will speed computing the inverse of  $A_B$ .

From the relationship of the original and new  $X_B$ ,  $x_i^{new} = x_i^{old} - x_{m+k}^{new} a_{ik}$ ,  $i = 1 \dots m, i \neq l$  and

$x_{m+k}^{new} = x_l^{old} / a_{lk}$ . Here  $a_{ik} = (B^{-1} P_{m+k})_i$ ,  $i = 1 \dots m$  ( $i$  refers to the  $i$ th element of the vector  $B^{-1} P_{m+k}$ ). We can see

$$D = (e_1, \dots, e_{l-1}, Ek, e_{l+1}, \dots, e_m) \text{ and } e_i = \begin{pmatrix} 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix}, \text{ and only element of } i \text{ row is 1, while the others}$$

are 0.  $Ek = (-a_{1k} / a_{lk}, \dots, -a_{(l-1)k} / a_{lk}, 1 / a_{lk}, -a_{(l+1)k} / a_{lk}, \dots, -a_{mk} / a_{lk})'$ .

This way, we can use the previous inverse matrix to calculate the new inverse matrix. Sposito (1989) gives a similar description of this method.

### 2.8.5 Applying method

For our case:

$$\text{Min } Z = CX$$

subject to:  $AX = b$  and  $X \geq 0$ .

Using artificial variables  $X_{av} = \begin{pmatrix} x_{n+1} \\ \dots \\ x_{n+m} \end{pmatrix}$ , the equation becomes  $AX + IX_{av} = b$ . Let

Con be a very big positive real number based on Big-M method. Then the objective function becomes  $Z = CX + Con * (1 \dots 1)X_{av}$ .

Based on the previous discussion, X are non-basic variables.  $A_B$  equals I. It is easy to compute. It's not needed to calculate the inverse matrix. But artificial variables are in the objective function. We must remove them from the objective function. If an artificial variable is removed from the basic variables, it will be removed from the objective function. This means the coefficient of the artificial variable becomes 0, not 1. After changing the coefficient of an artificial variable to 0, the artificial variable is in effect not present. When the optimum is reached, the coefficients of the artificial variables must be zero. Otherwise, there is no optimum.

### 2.8.6 An example

Minimize  $Z = x_1 - 3x_2 + 2x_3$

$$\text{subject to: } \begin{cases} 3x_1 - x_2 + 2x_3 = 7 \\ -2x_1 + 4x_2 = 12 \\ x_i \geq 0, i = 1, 2, 3 \end{cases}$$

Solution:

using artificial variables  $x_4$  and  $x_5$ , we can get an initial feasible solution. The question changes to:

minimize  $Z = x_1 - 3x_2 + 2x_3 + M * (x_4 + x_5)$

$$\text{subject to: } \begin{cases} 3x_1 - x_2 + 2x_3 + x_4 = 7 \\ -2x_1 + 4x_2 + x_5 = 12 \\ x_i \geq 0, i = 1, \dots, 5 \end{cases}$$

To remove the effects of the artificial variables, we set the coefficient M of the artificial variables in the objection function to a big real number, for example 100000.

**Iteration 1:**

$$C = (1 \quad -3 \quad 2 \quad 100000 \quad 100000), \quad b = \begin{pmatrix} 7 \\ 12 \end{pmatrix}, \quad A = \begin{vmatrix} 3 & -1 & 2 \\ -2 & 4 & 0 \end{vmatrix} = (P_1 \quad P_2 \quad P_3), \quad A_B^0 = I.$$

$x_4$  and  $x_5$  are the basic variables.  $A_N = A$ .

$$W = C_N - C_B A_B^{-1} A_N = (1 \quad -3 \quad 2) - (100000 \quad 100000) \begin{vmatrix} 3 & -1 & 2 \\ -2 & 4 & 0 \end{vmatrix}$$

$$= (1 \quad -3 \quad 2) - (100000 \quad 300000 \quad 200000) = (-99999 \quad -300003 \quad -199998)$$

So  $x_2$  is the entering variable. In the next step we will decide on the leaving variable.

$$A_B^{-1}b = \begin{pmatrix} 7 \\ 12 \end{pmatrix}, A_B^{-1}P_2 = \begin{pmatrix} -1 \\ 4 \end{pmatrix}. \text{ So the leaving variable is } x_5. a_{lk} = 4.$$

$$\text{So } E_k = (1/4, 1/4). \text{ We get } A_B^{1-1} = EA_B^{0-1} = \begin{vmatrix} 1 & 1/4 \\ 0 & 1/4 \end{vmatrix}.$$

### Iteration 2:

Now  $x_4$  and  $x_2$  are basic variables, and  $x_5$  is discarded.  $C = (1 \quad -3 \quad 2 \quad 100000)$ .

$$A_N = \begin{vmatrix} 3 & 2 \\ -2 & 0 \end{vmatrix}$$

$$W = C_N - C_B A_B^{-1} A_N = (1 \quad 2) - (100000 \quad -3) \begin{vmatrix} 1 & 1/4 \\ 0 & 1/4 \end{vmatrix} \begin{vmatrix} 3 & 2 \\ -2 & 0 \end{vmatrix}$$

$$= (1 \quad 2) - (250001.5 \quad 200000) = (-250000.5 \quad -199998)$$

So  $x_1$  is the entering variable.

$$A_B^{-1}b = \begin{pmatrix} 10 \\ 3 \end{pmatrix}, A_B^{-1}P_1 = \begin{pmatrix} 2.5 \\ -0.5 \end{pmatrix}. \text{ So the leaving variable is } x_4. a_{lk} = 2.5.$$

$$\text{So } E_k = (2/5, 1/5). \text{ We get } A_B^{2-1} = EA_B^{1-1} = \begin{vmatrix} 2/5 & 1/10 \\ 1/5 & 3/10 \end{vmatrix}.$$

### Iteration 3:

Now  $x_1$  and  $x_2$  are basic variables, and  $x_4$  is discarded.  $C = (1 \quad -3 \quad 2)$ .  $A_N = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$ .

$$W = C_N - C_B A_B^{-1} A_N = (2) - (1 \quad -3) \begin{vmatrix} 2/5 & 1/10 \\ 1/5 & 3/10 \end{vmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$= 2 - (-2/5) = 2.4.$$

So the optimal solution becomes:

$$X_B = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = A_B^{-1}b = \begin{pmatrix} 4 \\ 5 \end{pmatrix}, Z = C_B X_B = (1 \quad -3) \begin{pmatrix} 4 \\ 5 \end{pmatrix} = -11.$$

## 2.9 Nonlinear optimization

For most cases, there is a function  $f(x)$ , called the objective function, which belongs to  $C^2$ , meaning that the function  $f(x)$  has a second derivative. We want to find the minimum or maximum value of  $f(x)$ . We can describe this question as follows:

$$\min_x f(x)$$

Subject to:

$$x \in R^n$$

where  $R^n$  is the n-dimension real domain.

For the maximization question, we convert it to the minimization problem according to the following formulation:

$$\max_x f(x) = -\min_x (-f(x))$$

So we only need to solve the minimization question.

In this case, the variable  $x$  belongs to the n-dimension real domain. The number of dimensions may vary from 1 to n. This kind of minimization problem is called

**unconstrained optimization.**

If any constraints are applied to the variable  $x$ , we have the following situation:

$$\min_x f(x)$$

subject to:

The  $p$  equality constraints are:  $e_i(x) = 0$  for  $i=1,2,\dots,p$

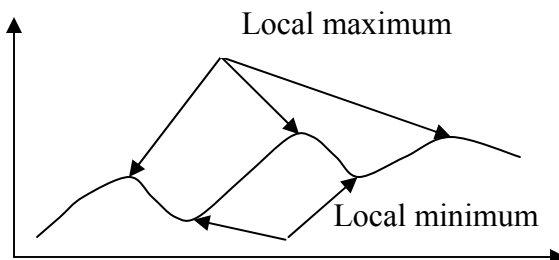
and the  $q$  inequality constraints are:  $w_j(x) \geq 0$  for  $j=1,2,\dots,q$ .

This kind of problem is called **constrained optimization.**

All points  $x$  satisfying all the constraints are feasible and all others are non-feasible. All feasible  $x$  form the feasible region. All non-feasible  $x$  form the non-feasible region. For unconstrained optimization, the feasible region is the real domain.

### 2.9.1 Local and global optimums

A local maximum is a point in the feasible region which is higher than all other points within its immediate vicinity, but not necessarily the whole feasible region. The global maximum is the maximum for the whole feasible region. The following figure illustrates local optimums:



From this figure, we can see the following points about the global and local optimums.

- There may be more than one local optimum for the function and their values perhaps are not the same.
- The global optimum must be a local optimum.
- A local optimum may be the global optimum.
- It is possible that there is more than one global minimum or maximum, if the function values are be same.

The global optimum is the best of all the local optimums and is the solution for our problem.

### 2.9.2 Classical theory of unconstrained optimization

Given a function  $f(x)$ , for vector  $x$ , assume all the first derivatives  $\frac{\partial f}{\partial x_i}$  exist at all points in the domain of  $f$ .

A **necessary** statement for a minimum of  $f(x)$  is:

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_2} = \dots = \frac{\partial f}{\partial x_n} = 0.$$

The condition “necessary” means that where the function is at a minimum, the equation holds. But this equation is not a sufficient condition.

A **sufficient** condition for a point to be a minimum of  $f(x)$  is that the second derivatives of function  $f(x)$  exist at the optimum point and  $D_i > 0$ .

$$D_i = \begin{vmatrix} \frac{\partial^2 f}{\partial x_1^2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_i} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_i \partial x_1} & \dots & \frac{\partial^2 f}{\partial x_i^2} \end{vmatrix}$$

Note: when the derivatives of the function  $f(x)$  are discontinuous, the classical theory is not fully applicable.

### 2.9.3 Finding a solution iteratively

Almost all numerical optimizations methods use iterative techniques. They start at an initial point  $x_0$  and proceed by generating a sequence of points  $x_1, \dots, x_m$  (each  $x_i$  is an  $n$ -

dimension vector). Let  $f(x_{i+1}) \leq f(x_i)$ . Then, the minimum of  $f(x)$  will be approached more closely with each iteration. Clearly, the choice of  $x_i$  is very important.

Defined by  $x_{i+1} = x_i + d_i s_i$ ,  $d_i$  is a direction vector for finding the next  $x$  and  $s_i$  is the step size or distance to move. Here, a suitable choice of direction  $d_i$  is very important. How to search for the next  $x$  is an important issue. Typically, methods are classified into two classes: direct search and gradient methods.

#### **2.9.4 Search methods: direct and gradient**

Direct search methods don't require the explicit evaluation of any derivatives of the function, but rely solely on values of the objective function  $f(x)$  and information gained from earlier iterations. Some use function values to obtain numerical approximations of the derivatives.

Gradient methods select the direction using the values of the derivatives of the function  $f(x)$ . Usually, the first order derivatives are used by these methods.

#### **2.9.5 Converting constrained to unconstrained optimization**

For constrained optimization problems, it can be useful to make use of unconstrained optimization methods. So converting to an unconstrained optimization problem is the first task. Many methods have been developed for transforming the optimization problem. The following methods are widely used:

1. Transfer functions
2. Lagrangian multipliers
3. Penalty functions

##### **2.9.5.1 Transfer functions**

Its basic idea is to extend the restricted feasible region to the whole real domain. For example, to minimize  $f(x)$ , subject to  $x > a$ , we can define a new variable  $y$ . Let

$$x = a + y^2$$

Using this equation, we can convert  $f(x)$  to  $f(y)$ , and then minimize  $f(y)$ . Here variable  $y$  doesn't have any restriction. So this is now an unconstrained optimization problem.

### 2.9.5.2 Lagrangian multipliers

This is a very common method for transforming optimization problems. If a minimization problem has many equality constraints

$$e_i(x) = 0 \text{ for } i=1,2,\dots,p$$

define a new objective function to minimize with a new variable  $\lambda$

$$h(x, \lambda) = f(x) + \sum_{j=1}^p \lambda_j e_j(x).$$

For the first derivatives of this function,

$$\frac{\partial h(x, \lambda)}{\partial x_i} = \frac{\partial f(x)}{\partial x_i} + \sum_{j=1}^p \lambda_j \frac{\partial e_j(x)}{\partial x_i} = 0$$

$$\frac{\partial h(x, \lambda)}{\partial \lambda_i} = e_j(x) = 0$$

The solution will satisfy the constraints  $e_i(x) = 0$ .

For the inequality constraints

$$w_j(x) \geq 0 \text{ for } j=1,2,\dots,q$$

we can introduce new variables called slack variables,  $x_{n+1} \dots x_{n+q}$ . Let

$$w_j(x) = x_{n+j}^2 \geq 0.$$

Now we can transform the inequality into equality:

$$w_j(x) - x_{n+j}^2 = 0$$

So using this method, we can handle the constrained optimization problem.

### 2.9.5.3 Penalty functions

The basis for the penalty function method is to define a new objective function like the following:

$$h(x) = f(x) + p(c(x)),$$

where  $f(x)$  is the original objective function, and  $p(c(x))$  is the penalty function based on the equality and inequality constraints.

For a minimization problem, the main point is to choose the penalty function to make sure that it is zero for all feasible points and is very high for all non-feasible points. Then, the minimum of  $h(x)$  is equivalent to the minimum of  $f(x)$ .



### 2.9.6 Our case

For our problems, the optimization question is defined as follows:

Find the minimum and maximum of function

$$f(x, y) = \sum_i x_i p_{xi} \sum_j y_j p_{yj} + \rho \sqrt{(\sum_i x_i^2 p_{xi} - (\sum_i x_i p_{xi})^2)(\sum_j y_j^2 p_{yj} - (\sum_j y_j p_{yj})^2)}$$

subject to:

$$l \leq x \leq u$$

$$s \leq y \leq p$$

where  $l = (l_1 \dots l_n)$ ,  $u = (u_1 \dots u_n)$ ,  $s = (s_1 \dots s_m)$  and  $p = (p_1 \dots p_m)$ .  $l_i$ ,  $u_i$ ,  $s_i$  and  $p_i$  are real numbers, not infinity. This kind of question is called box-constrained optimization or bound-constrained optimization.

We only discuss the minimization problem. For maximum problems, we can use the previous formulation to convert them to minimization problems.

Next, we need to convert the problem to an unconstrained optimization. Let use the three methods introduced previously.

#### 2.9.6.1 Transfer function

The constraints for variable  $x$  and  $y$  are  $l \leq x \leq u$ , and  $s \leq y \leq p$ . This means that  $x$  lies between  $l$  and  $u$  and  $y$  lies between  $s$  and  $p$ . so we need to introduce a new variable to replace  $x$  and make sure  $x$  satisfies the constraint. Defining

$$x = l + (u - l) \sin^2 u, \text{ and } y = s + (p - s) \sin^2 v$$

we will get the new objective function  $f(u, v)$ .

For this function,  $y$  belongs to the whole real domain, so it is unconstrained. But this function is very complicated. If you want to use the first derivative to get the solution, it is tricky because  $y$  has many solutions.

#### 2.9.6.2 Lagrangian multipliers

For  $l \leq x \leq u$ ,  $s \leq y \leq p$ , we can convert to:

$x - l \geq 0$ ,  $u - x \geq 0$ ,  $y - s \geq 0$ , and  $p - y \geq 0$ . Using the previous methods, we can get a new objective function. But this method introduces many slack variables and equalities. To solve these equalities is not easy work. It needs much CPU time to compute and it is also very complicated.

### 2.9.6.3 Penalty function

We will design a suitable penalty function. Based on the constraints, we introduce this penalty function:

$$p(x) = \lambda * \max(0, l_1 - x_1, x_1 - \mu_1, \dots, l_n - x_n, x_n - \mu_n, s_1 - y_1, y_1 - p_1, \dots, s_m - y_m, y_m - p_m)$$

Here,  $\lambda$  will be chosen as a very large positive real number. So the new objective function is

$$h(x, y) = f(x, y) + p(x, y).$$

From this function, we can see that if  $l \leq x \leq u$ , and  $s \leq y \leq p$ , then  $x$  and  $y$  belong to the feasible region and  $h(x, y)$  equals  $f(x, y)$ , but if constraints are violated,  $h(x, y)$  will become very large, clearly far from the minimum value.

### 2.9.6.4 Search method

Our objective function has a good attribute; both the first derivatives and second derivatives exist. So gradient search (Luenberger, 1984, pp. 384) can be freely applied to our case. And generally speaking, gradient searching methods provide efficient direction information in searching for the next  $x$ . In view of the previous discussion, gradient search is used to our case.

### 2.9.6.5 Solution

Find the minimum value of function  $f(x)$ , stated by

$$\text{Min } f(x, y) = \sum_i x_i p_{xi} \sum_j y_j p_{yj} + \rho \sqrt{(\sum_i x_i^2 p_{xi} - (\sum_i x_i p_{xi})^2)(\sum_j y_j^2 p_{yj} - (\sum_j y_j p_{yj})^2)}$$

subject to:

$$l \leq x \leq u$$

$$s \leq y \leq p$$

where  $l = (l_1 \dots l_n)$ ,  $u = (u_1 \dots u_n)$ ,  $s = (s_1 \dots s_m)$  and  $p = (p_1 \dots p_m)$ .  $l_i$ ,  $u_i$ ,  $s_i$  and  $p_i$  are real numbers, not infinity.

We use a penalty function to convert this problem to an unconstrained problem. The new objective function  $h(x, y)$  is constructed as:

$$h(x, y) = f(x, y) + \lambda * \max(0, l_1 - x_1, x_1 - \mu_1, \dots, l_n - x_n, x_n - \mu_n, s_1 - y_1, y_1 - p_1, \dots, s_m - y_m, y_m - p_m) S$$

o the problem is to find the minimum value for function  $h(x, y)$ . For this unconstrained

optimization problem, the iterative technique is adopted. First, we define some terms:

$$\bar{x} = \sum_i^n x_i p_{xi}$$

$$\bar{y} = \sum_j^m y_j p_{yj}$$

$$D_x = (\sum_i x_i^2 p_{xi} - (\sum_i x_i p_{xi})^2)$$

$$D_y = (\sum_j y_j^2 p_{yj} - (\sum_j y_j p_{yj})^2)$$

$$D_{xy} = D_x * D_y.$$

Now we get the first derivative of  $h(x,y)$  through  $f(x,y)$  and penalty function  $p(x,y)$ .

$$\frac{\partial f}{\partial x_i} = p_{xi} \bar{y} + \rho * D_{xy}^{-1/2} * \frac{1}{2} * D_y * (2x_i p_{xi} - 2\bar{x} p_{xi})$$

$$\frac{\partial f}{\partial y_j} = p_{yj} \bar{x} + \rho * D_{xy}^{-1/2} * \frac{1}{2} * D_x * (2y_j p_{yj} - 2\bar{y} p_{yj})$$

$$\frac{\partial f}{\partial \rho} = D_{xy}^{1/2}$$

$$\frac{\partial p}{\partial x_i} = \begin{cases} 0 & \text{others} \\ \lambda & x_i - u_i = \max(0, l_1 - x_1, x_1 - \mu_1, \dots, l_n - x_n, x_n - \mu_n, s_1 - y_1, y_1 - p_1, \dots, s_m - y_m, y_m - p_m) \\ -\lambda & l_i - x_i = \max(0, l_1 - x_1, x_1 - \mu_1, \dots, l_n - x_n, x_n - \mu_n, s_1 - y_1, y_1 - p_1, \dots, s_m - y_m, y_m - p_m) \end{cases}$$

$$\frac{\partial p}{\partial y_j} = \begin{cases} 0 & \text{others} \\ \lambda & y_{ji} - p_j = \max(0, l_1 - x_1, x_1 - \mu_1, \dots, l_n - x_n, x_n - \mu_n, s_1 - y_1, y_1 - p_1, \dots, s_m - y_m, y_m - p_m) \\ -\lambda & s_j - y_j = \max(0, l_1 - x_1, x_1 - \mu_1, \dots, l_n - x_n, x_n - \mu_n, s_1 - y_1, y_1 - p_1, \dots, s_m - y_m, y_m - p_m) \end{cases}$$

Along the direction determined by the derivatives, the next  $x$  and  $y$  are defined.

Through iteration, the numerical solution can be found.

Next, finding the maximum value of function  $f(x)$ ,

Max  $f(x, y)$

subject to:

$$\begin{aligned} l &\leq x \leq u, \\ s &\leq y \leq p. \end{aligned}$$

Based on the formulation  $\max_x f(x, y) = -\min_x (-f(x, y))$ , we can transform this problem to:

$$\begin{aligned} & \text{Min } -f(x, y) \\ & \text{subject to:} \\ & \quad l \leq x \leq u, \\ & \quad s \leq y \leq p. \end{aligned}$$

Using the previous method, we can get the minimum value  $f_{\min}$ , and negate to get the maximum value of  $f(x)$ ,  $-f_{\min}$ .

### 3 Enhancement of functions

This version of Statool removes certain important limitations existing in the previous version. The following extensions had to be developed in order to apply Statool to the problems we wanted to solve.

- Use of the transportation method to speed linear programming
- Cascading operations to support more than two variables
- Relational operations
- Evaluation of  $f(x,y)$  for monotonic functions  $f$

#### 3.1 *Transportation method*

In the previous version, only the standard simplex method is provided to solve linear programming. The speed of this method is slower than that of the transportation simplex method.

##### 3.1.1 Background on the transportation simplex method

Many companies need to determine how to optimally transport goods from different warehouses to different destinations. Isomorphic problems are found in other situations unrelated to transportation, such as the assignment problem and production scheduling. Hillier (2001) gave detailed information about such applications.

##### 3.1.1.1 Model

In general, this kind of problem involves 2 different types of location: sources and destinations. Sources supply something and destinations accept resource. Costs for transferring resources between each source and destination may be different. The aim is to minimize the total cost to transfer resource from these sources to those destinations. In most cases, the total supply for all sources is equal to the total demand for all destinations. If we have  $M$  sources,  $N$  destinations, the supply at source  $i$  is  $S_i$ , and the demand at destination  $j$  is  $D_j$ , we get the equation  $\sum_{i=1}^M S_i = \sum_{j=1}^N D_j$ . Let  $C_{ij}$  be the unit cost of moving resources from

source  $i$  to destination  $j$ . This table displays the relationship between sources and destinations.

**Table 3.1. Parameter table for transportation model.**

Source	Cost per unit distributed					Supply
	Destination					
	1	2	3	.....	N	
1	C11	C12	C13		$C_{1N}$	S1
2	C21	C22	C23		$C_{2N}$	S2
.....						
M	$C_{M1}$	$C_{M2}$	$C_{M3}$		$C_{MN}$	$S_M$
Demand	D1	D2	D3		$D_N$	

We can describe this mode as a standard linear programming problem.

$$\min Z = \sum_{i=1}^M \sum_{j=1}^N C_{ij} x_{ij}$$

Subject to:

$$\sum_{j=1}^N x_{ij} = S_i \quad \text{for } i=1 \dots M$$

$$\sum_{i=1}^M x_{ij} = D_j \quad \text{for } j=1 \dots N$$

$$\text{and } x_{ij} \geq 0 \quad \text{for all } i \text{ and } j$$

If total supply is not equal to total demand, it is called an unbalanced model. For these cases, we can use dummy sources or destinations to make the model balance. If total supply is greater than total demand, we can make up dummy destinations to demand extra resources and set the unit cost from each source to any dummy destinations to be very small. This way, extra resources will be transferred to dummy destinations. If total supply is less than total

demand, we make up some dummy sources and set unit cost from each dummy source to any destinations very large. If these unit costs are really large, no destination will want to get resources from these dummy sources. So the solution will be for resources from actual sources rather than dummy sources.

### 3.1.1.2 Solution

The Transportation problem is a special type of linear programming. We can use general methods for linear programming such as the simplex method. If the simplex method is used, the simplex tableau will be complex and consists of  $M+N+1$  rows and  $(M+1)(N+1)$  columns. To handle this big table, you will need a lot of computation.

As a special type of linear programming problem, there is an efficient method called the transportation simplex method to handle it. This method uses a tableau, but it only has  $M$  rows and  $N$  columns. You don't need to use artificial variables to get an initial solution. It has just  $M+N-1$  basic variables (not  $M+N$ ), so a degree of freedom will be removed.

To solve transportation problems, generally two steps are necessary.

**Step One:** Initialization to get an initial basic feasible (BF) solution. There are 3 common methods for this step.

- Northwest corner rule
- Russell's approximation method
- Vogel's approximation method

Russell and Vogel's methods consider costs in generating an initial solution. The solutions are better than for the Northwest corner method. Hillier and Lieberman (2001) clearly compares these three methods.

**Step two:** Optimality testing. In this step, every solution is a feasible solution. Our aim is to find the best solution. It has a loop to do the following work.

- Get the two variables  $u_i$  and  $v_j$  from each basic variable's equation ( $C_{ij}=u_i+v_j$ ).
- Calculate the related cost  $CC_{ij}$  of each non-basic variable according to  $CC_{ij}=C_{ij}-u_i-v_j$ .
- Get the entering non-basic variable, the one with the minimum  $CC_{ij}$  of all non-basic variables with negative  $CC_{ij}$ .

- Determine whether the solution is optimal. If all  $CC_{ij}$  are not less than 0, the solution is optimal.
- Get the leaving basic variable. This is done in a loop whose calculations use the entering non-basic variable and other basic variables. This loop identifies the cell whose assigned flow is the minimum and whose order to the entering cell is odd. This cell will be the leaving variable.
- Adjust the flow of the loop. For all the cells adjacent to the entering cell or another odd distance from it in the loop, subtract the minimum flow and for all cells an even distance, add the minimum.
- Get the new basic variable set. Marking the entering cell basic variable and the leaving cell non-basic variable. Begin the loop again from step 1.

### 3.1.2 Exceptions in finding the initial solution

Handling the exception of degeneracy can be very important in finding the initial solution in a transportation simplex problem. Degeneracy means there are not enough basic variables in the initial feasible solution. For example, there are 5 basic variables for  $2 \times 3$  tables. In fact, maybe only 4 variables are found for some initialization methods for some problems. This situation occurs where there are too many choices for which ones are basic. In the previous initialization methods, the northwest corner method doesn't have this kind of problem. This method always can find enough basic variables although values of some of them may be zero. But Russell's method will have this kind of problem for some cases. Usually, the initial solution found by Russell's method is closer to the optimal solution than that found by the northwest corner method. So computing time is less for Russell's method. Therefore, there is a tradeoff.

### 3.1.3 Adaptation to the unknown dependency case

For the unknown dependency case, the marginal distribution table for variables  $X$  and  $Y$  is really a transportation tableau. Here you can consider  $X$  as the sources and  $Y$  as the destinations. The total supply is 1 and total demand is also 1. Next we use an example to illustrate this situation.

**Example:**



X distribution:  $P([0,1]) = 0.2$ ,  $P([1,2]) = 0.2$ ,  $P([2,3]) = 0.2$ ,  $P([3,4]) = 0.4$ .

Y distribution:  $P([1,2]) = 0.25$ ,  $P([2,3]) = 0.25$ ,  $P([3,4]) = 0.2$ ,  $P([4,5]) = 0.3$ .

Consider  $X+Y$  for the case of unknown dependency. We get the marginal distribution table next:

**Table 3.2. Marginal distribution**

X \ Y	[0,1]	[1,2]	[2,3]	[3,4]	Prob.
[1,2]	[1,3] p11	[2,4] p12	[3,5] p13	[4,6] p14	0.25
[2,3]	[2,4] p21	[3,5] p22	[4,6] p23	[5,7] p24	0.25
[3,4]	[3,5] p31	[4,6] p32	[5,7] p33	[6,8] p34	0.2
[4,5]	[4,6] p41	[5,7] p42	[6,8] p43	[7,9] p44	0.3
Prob.	0.2	0.2	0.2	0.4	1

Our question is how to assign the distribution to p11 ... to p44 to give some subset a maximized probability. E.g. to find the upper bound for  $X+Y$  at 1 (the previous chapter discussed finding the subset), we get the linear programming problem

Max  $f = p11$

subject to:

$$p11 + p12 + p13 + p14 = 0.25$$

$$p11 + p21 + p31 + p41 = 0.2$$

.....

To find the upper bound for the CDF at 2, we get the problem

$$\text{Max } f = p11 + p12 + p21$$

subject to:

$$p11 + p12 + p13 + p14 = 0.25$$

$$p11 + p21 + p31 + p41 = 0.2$$

.....

For every point in the support of the result distribution, we will get a linear programming problem. Through solving these problems, the upper bound of the CDF will be gotten.

The low bound of the CDF is found similarly to the upper bound. To speed the solutions, we use the transportation method to solve these linear programming problems.

From the previous example, we can see these linear programming problems use transportation tables. The main difference is to maximize the value of the objective function rather than the minimize as in real transportation problems. To solve these problems, we can use negation to transform Max to Min. The  $C_{ij}$  are very important in transforming the problems. For the objective function, we have to let  $C_{ij}$  be 1, 0, or -1. We need to transform Max to Min, so we must use  $C_{ij}=-1$  for all items that will contribute to the objective function, with others zero. For the previous 2 cases we will get:

$$\text{Min } -f = -p_{11}$$

subject to:

$$p_{11} + p_{12} + p_{13} + p_{14} = 0.25$$

$$p_{11} + p_{21} + p_{31} + p_{41} = 0.2$$

$$C_{11} = -1, \text{ other } C_{ij} = 0$$

$$\text{Min } -f = -p_{11} - p_{12} - p_{21}$$

subject to:

$$p_{11} + p_{12} + p_{13} + p_{14} = 0.25$$

$$p_{11} + p_{21} + p_{31} + p_{41} = 0.2$$

$$C_{11} = C_{12} = C_{21} = -1, \text{ other } C_{ij} = 0$$

Thus we have a way to transform an unknown dependency case to a transportation problem. It includes two steps:

- Get the transportation table from the marginal distribution table
- Set the cost attribute for cells contributing to the objective function to  $-1$ , and other cells' cost to zero.

Because the balance of supply and demand is a basic requirement for the transportation problem, we must keep marginal sum of X and Y equal to 1 and the same for Y.

### 3.1.4 Test result:

Consider an example:

$$X, p([0, 0.333]) = 0.2, p([0.333, 0.667]) = 0.4, p([0.667, 0.999]) = 0.4$$

$$Y, p([0, 0.5]) = 0.5565437, p([0.5, 1]) = 0.4434564.$$

Consider  $X+Y$  under the unknown dependency condition.

**Table 3.3. Lower bound.**

Result interval	Simplex	Transportation
$[-0.25, 0.833]$	-1.490116E-08	0
$[0.833, 1.167]$	-1.490116E-08	0
$[1.167, 1.333]$	0.1565436	0.1565437
$[1.333, 1.5]$	0.2	0.1999999
$[1.5, 1.667]$	0.5565436	0.5565436
$[1.667, 2]$	0.6	0.6
$[2, 2.25]$	1	1

**Table 3.4. Upper bound.**

Result interval	Simplex	Transportation
$[-0.25, 0]$	0	0
$[0, 0.333]$	0.2	0.2
$[0.333, 0.5]$	0.5565437	0.5565437
$[0.5, 0.667]$	0.6	0.6
$[0.667, 0.833]$	0.7565437	0.7565438
$[0.833, 1.167]$	1	1
$[1.167, 2.25]$	1	1

For this example, we got almost the same answer for both methods.

### 3.2 Cascading operations

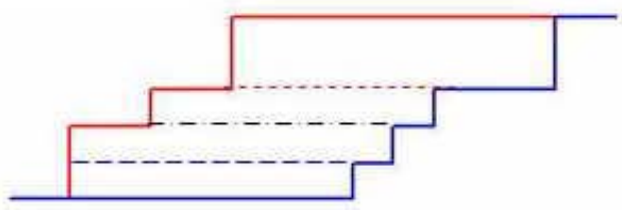
Previous Statool software only supported binary operations (two operands). But in real applications, there are often over 2 operands to be calculated, for example,  $x+y+z$ ,  $\text{Max}(x,y,z)$ , etc.

Association is how to solve this question. E.g. for  $x+y+z$ , we can first calculate  $x+y$ , and save the result to temporary variable  $w=x+y$ , then calculate  $w+z$ . This way, we can get  $x+y+z$ . But there is a constraint that this kind of operation must support association and commutation. For example, you can first calculate  $x+y$  or  $y+z$ , for  $x+y+z$ .

In Statool, we would get the CDF envelopes for the result of two variables' operation under unknown dependence. So we can solve this problem if we can convert CDF envelopes to a set of intervals and associated probabilities.

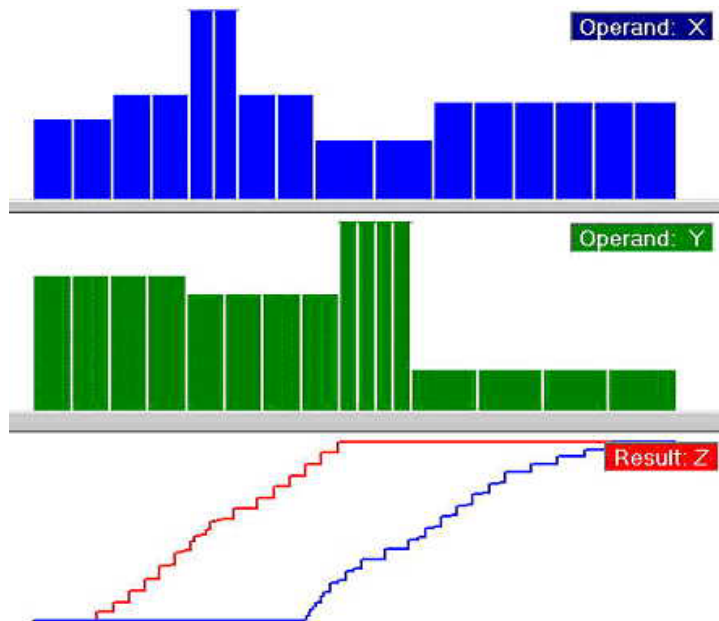
### 3.2.1 Solution

We can transform upper and lower envelopes into a set of intervals and associated probabilities. The probability of each envelope is its top-to-bottom height. For example, four intervals will be gotten from the following CDF envelopes.



**Figure 3.1. Convert CDF to IDF.**

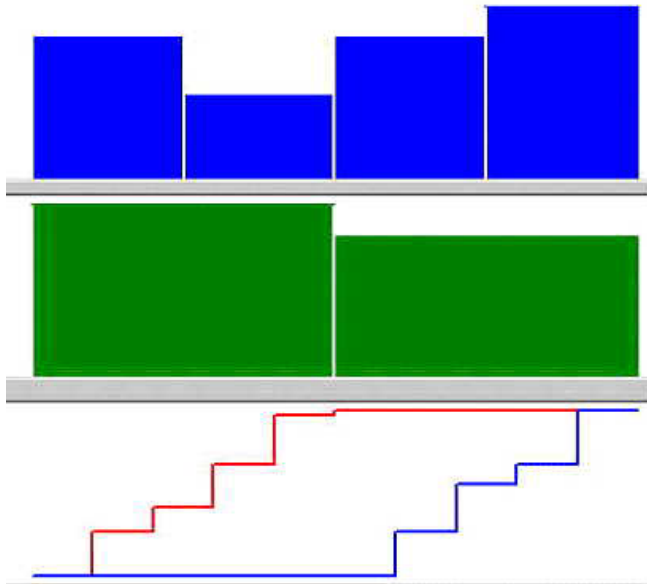
This method is implemented in Statool using VB. Now both CDF and IDF data format can be saved or displayed. From the following figures, we can now use the results of one operation as input to the next.



**Figure 3.2. Result for operation.**

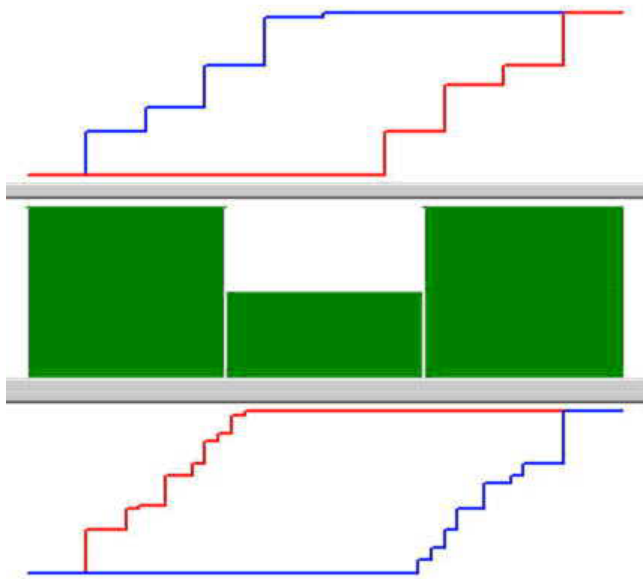
The above figure shows the CDF envelopes resulting from an operation on two variables.

The following figures show the procedure to calculate using multiple operands (e.g.  $x+y+z$ ).



**Figure 3.3. Result for  $x+y$ .**

First, we get the result of  $x+y$ . Result is shown in the 3<sup>rd</sup> panel.



**Figure 3.4. Result for  $x+y+z$ .**

Then we load the result of  $x+y$  as a new operand (top panel above) and operate on it and  $z$ , which is shown in the middle panel. The bottom panel shows  $x+y+z$ .

### 3.3 Relational operations

Relational operations are used to describe the relationship between two operands.

This version of Statool supports 4 relational operations:  $>$ ,  $>=$ ,  $<$ , and  $<=$ .

#### 3.3.1 Relational operations on intervals

Consider two real numbers  $x$  and  $y$ . We define the interval value to describe the relationship between  $x$  and  $y$ . The value  $[0,0]$  indicates the relationship is false. The value  $[1,1]$  indicates the relational operation is true. The value  $[0,1]$  means the value of the relational operation is not determined or it's uncertain.

For interval number  $A$ ,  $A$ -left means the left (or low) bound of interval value  $A$ , and  $A$ -right means the right (or high) bound of it. Now consider two interval numbers  $A$  and  $B$ .

$$A > B = \begin{cases} [1,1] & A - \text{left} > B - \text{right} \\ [0,0] & A - \text{right} \leq B - \text{left} \\ [0,1] & \text{otherwise} \end{cases}$$

$$A \geq B = \begin{cases} [1,1] & A - \text{left} \geq B - \text{right} \\ [0,0] & A - \text{right} < B - \text{left} \\ [0,1] & \text{otherwise} \end{cases}$$

$$A < B = \begin{cases} [1,1] & A - \text{right} < B - \text{left} \\ [0,0] & A - \text{left} \geq B - \text{right} \\ [0,1] & \text{otherwise} \end{cases}$$

$$A \leq B = \begin{cases} [1,1] & A - \text{right} \leq B - \text{left} \\ [0,0] & A - \text{left} > B - \text{right} \\ [0,1] & \text{otherwise} \end{cases}$$

#### 3.3.2 Relational operations on random variables

Consider 2 random variables  $X$  and  $Y$ . We consider the probability of  $X > Y$ ,  $P\{X > Y\}$ .

According to the DEnv algorithm, random variables  $X$  and  $Y$  are split into intervals which are assigned probabilities. Therefore, operation  $X > Y$  is transformed into a series of interval operations. Here is an example to show how to handle this.

**Table 3.5. Distribution for X and Y.**

X \ Y	[0,1]	[1,2]	[2,3]	Prob.
[1,2]	p11	p12	p13	0.25
[2,3]	p21	p22	p23	0.5
[3,4]	p31	P32	p33	0.25
Prob.	0.5	0.25	0.25	1

Consider the relational operation  $X > Y$ . It is transformed into an interval relational operation between intervals of X and intervals of Y. for example, the result of  $[0,1] > [1,2]$  is  $[0,0]$ , so  $[0,0]$  will be put into cell p11. similarly,  $[0,1]$  will be put into cell p12. Finally, we get the following result:

**Table 3.6. Interval value for relational operation.**

X \ Y	[0,1]	[1,2]	[2,3]	Prob.
[1,2]	$[0,0]$ p11	$[0,1]$ p12	$[0,1]$ p13	0.25
[2,3]	$[0,0]$ p21	$[0,0]$ p22	$[0,1]$ p23	0.5
[3,4]	$[0,0]$ p31	$[0,0]$ P32	$[0,0]$ p33	0.25
Prob.	0.5	0.25	0.25	1

Based on the DEnv algorithm, we can now get the probability for  $X > Y$ . It is clear that all cells whose interval bounds include 1 are consistent with  $X > Y$ . To get the maximum value of  $P\{x > y\}$ , the sum of all cells including 1 will be maximized. For this case, maximize  $(p12+p13+p23)$ . To get the minimum value of  $P\{x > y\}$ , the sum of all cells with the value  $[1,1]$  will be minimized. All cells whose value is  $[0,1]$  will be discarded since for them, maybe  $x \leq y$ .

In summary, the value of each cell should be one of  $[0,0]$ ,  $[0,1]$ , and  $[1,1]$ . Here  $[0,0]$  means this relationship doesn't hold. The value  $[0,1]$  means this relationship is not certain. The value  $[1,1]$  indicates this relationship must hold. To get the maximum value, maximize

the sum of all cells whose bound include 1. To get the minimum value, minimize the sum of all cells whose values are  $[1,1]$ .

### 3.4 Complex expressions

The previous Statool only supported the basic arithmetic operations  $+$ ,  $-$ ,  $*$ , and  $/$ . It is more useful to be able to calculate any arithmetic expression. User should be able to input the expression desired. To solve this problem, Statool needed an arithmetic expression editor to provide the functionality to input an arithmetic formula. We decided to support expressions using arithmetic operators  $+$ ,  $-$ ,  $*$ , and  $/$  and also to support association through using  $( )$ .

#### 3.4.1 Expression editor

To implement this expression editor, first, a grammar definition of allowed expressions was written. This grammar is context-free. The following grammar describes arithmetic expressions that Statool supports:

```
<expression>::=<term> | <term> + <expression> | <term> - <expression>
<term>::=<factor> | <factor> * <term> | <factor> / <term>
<factor>::=( <expression> ) | <number> | <variable>
<number>::= <integer> | <integer>.<integer>
<integer>::=<integer>|v
<variable>::=x|y
```

Here v indicates the numbers from 0 to 9.

Based on this grammar, arithmetic expressions such as  $(a*X+b*Y)/(c*X+d*Y)$  are allowed. Parsing is a good method to generate a parse tree: a diagram of the complete grammatical structure of the string being parsed. For this case, it is not very complex. Every operator needs two operands.  $( )$  will increase the priority of operation. So, expression tables could be used. In such a table, the operands and operators will be recorded. Every row describes an operator. Software will analyze the input string, and generate the expression table according to the order of calculations. Then using this table, the result can be calculated.

Statool software now only supports 2 random variables, X and Y. When an expression is input, variable names must use the symbols X and Y. The expression editor will check the input expression after the user confirms the input. If this expression is not allowed,



error information will be displayed and reason also will be listed. Here is a figure showing the expression editor.

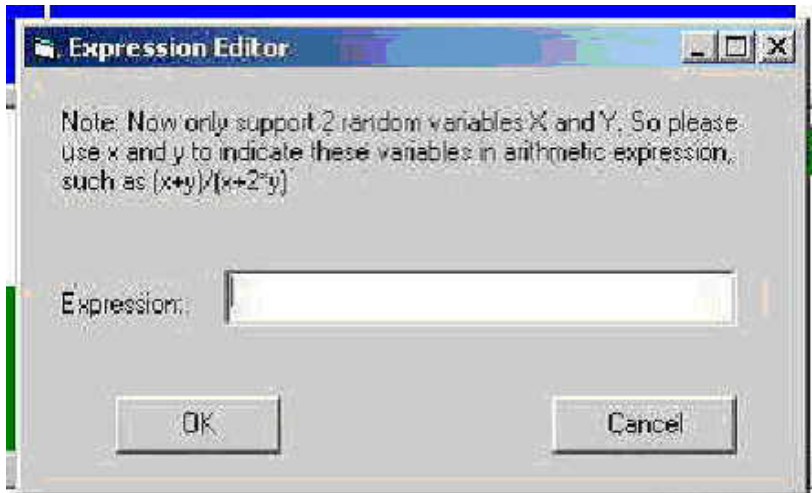


Figure 3.5. Expression editor.

From the following figure, we can see the error information and reason for a bad input expression.

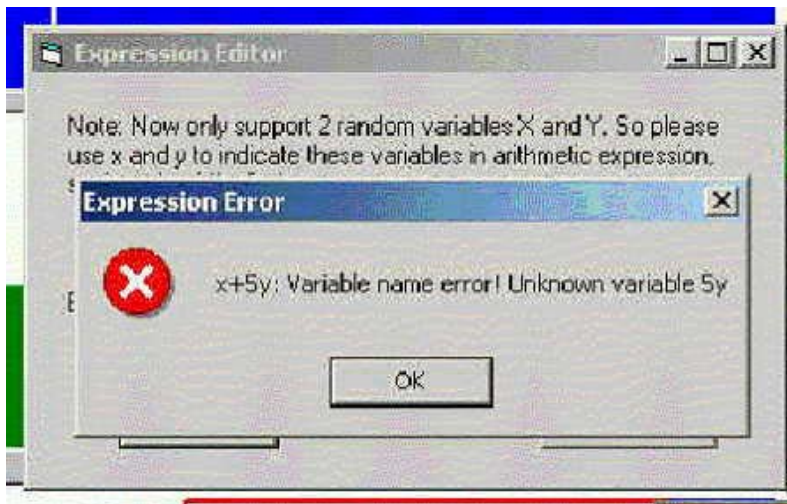


Figure 3.6. Error information for experssion editor.

### 3.4.2 Limitations on evaluating expressions

This expression editor can't handle division by 0 since the expression evaluator doesn't know how to evaluate the value of such expressions. Therefore, operands X and Y can't include zero in their support if the user want to use the expression editor.

### 3.4.3 Excess width in expressions

A typical expression is  $P(x,y) = f(x,y)/g(x,y) = (aX+bY)/(cX+dY)$ . For this kind of expression, there is excess width in interval calculations. The reasons include that a random variable is cited more than one time in the expression. To solve the problem, it is necessary to remove excess width in calculating this type of expression.

#### 3.4.3.1 Removing excess width

The easiest way to handle it is to simplify the expression such that each random variable is cited only one time. It is a quick way to handle this question. But it is a very restrictive constraint. Many expressions can't be simplified to meet this kind of condition.

For some expressions, we can use another way to remove excess width. It is to use the low and high bounds of the interval operands to calculate the expression. Then from these calculated values, the result bound is determined. For two variables, there are four combinations of bounds. So 4 candidate result values are obtained. We select the minimum of the 4 as the low bound of the result interval, and the maximum of the 4 as the high bound of the result interval. Let us see an example:

Suppose:  $x = [1,2]$ ,  $y = [2,3]$   $F(x,y) = (8.4x + 7.2y)/(0.04x + 0.02y)$ .

First: let  $x=1$ ,  $y=2$ , calculate  $F(x,y)$ , and we get the value 285.

Second: let  $x=1$ ,  $y=3$ , calculate  $F(x,y)$ , and we get the value 300.

Third let  $x=2$ ,  $y=2$ , calculate  $F(x,y)$ , and we get the value 260.

Finally, let  $x=2$ ,  $y=3$ , calculate  $F(x,y)$ , and we get the value 274.3.

So, the interval for  $F(x,y)$  is  $[260,300]$ .

If we calculate the expression based on the intervals for  $x$  and  $y$ , we will get the interval  $[162.9, 480]$ . It is obvious that result interval has excess width. So we can use this method to remove excess width for this expression.

#### 3.4.3.2 Limitation of the method for removing excess width

Although the method of selecting the min and max value to get the result bound works for our case, there are limitations to this method. When the expression is monotonic, the method can handle excess width. In addition, the denominator of the expression can not

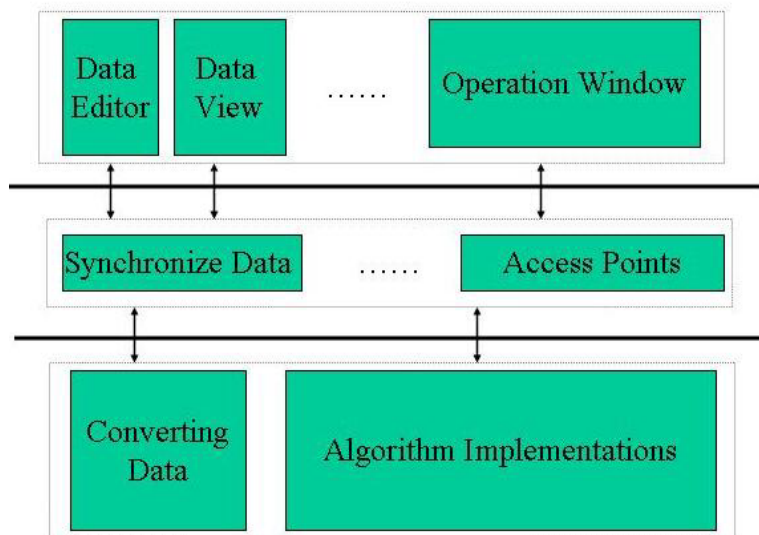
include the zero, that is, zero is not in the support of  $g(x,y)$ , otherwise this expression can't be calculated.

## 4 Software architecture

This chapter describes the architecture of “Statool”. First an overview of this software is given. Then the components of “Statool” are described.

### 4.1 Overview

Layer design is widely used for software development. It provides a clear description of software architecture and makes it easily understood. It is also suitable for implementation and maintenance of distributed computing software. Figure 4.1 shows the general layer architecture of “Statool.” This version was developed for a Microsoft Windows platform. Layer design is helpful for porting to other platforms.



**Figure 4.1. Architecture.**

Statool consists of 3 levels layers as shown in the previous figure. The first level, the application layer, is the user interface layer, which is in charge of interaction with the user such as receiving the user settings and displaying the results of operations. The middle level transforms user inputs to fit the underlying algorithms. This layer can be called the logical layer. The low level, the computing layer, implements the specified algorithms. It can run in the background and be provided as dynamically linked library.

Since Microsoft operating systems are widely used in the world, the platform for this software is the Microsoft windows series, such as Windows 98 and Windows 2000. But the

primary platform is the Windows 2000 family. The software will be fully tested on this platform. For other windows platforms, it won't be fully tested.

For the current software, the user interface and the logical layer were developed using Visual Basic. Visual Basic is a very good rapid development tool. Since the user interface is not designed to run in a Web browser currently, it was developed together with the logical layer component using Visual Basic. It may be useful to move the user interface to a Web browser platform in the future because the browser / server architecture is so popular. The computing layer involves a lot of computing, so speed is a key issue. Therefore, this layer was developed using Visual C++.

This software can be run on any machine in which windows 2000 is installed. There is no other requirement for hardware. But as previous noted, this software will consume a lot of computing and memory resource. If you solve bigger problems, a configuration should have at least 256M memory and a Pentium III at 1000 Mhz.

## **4.2 Input/output**

The file is the primary way to exchange and save the data for this software. Graphs are used to show probability distribution functions (PDFs) and cumulative distribution functions (CDFs) of the data. Data lists can show the exact information representing a random variable.

In this software, 4 kinds of files are used to describe random variables. One is called a probability distribution file (PDF), which is a list of intervals and probabilities and, and discretizes a probability density function. The second is called a cumulative distribution file (CDF), which is used to describe the envelopes of the cumulative distribution. The third kind is called an intermediate distribution file (IDF), which is like a .PDF file but the intervals can overlap. Although its interval can be overlapped, the sum of their probabilities should be equal to 1. Otherwise, this file is invalid. The last one is the sample data file (SMP). It is a special type of file. It is used to show the probability distribution of sample data. Only IDF and SMP files are used to input or output distributions. PDF and CDF files are mainly used internally. Output displays are plotted based on these 2 types of files.

Both IDF and SMP are text files. They are composed of 2 parts: a control line and data lines. The control line is the first line and includes 2 items: the number of intervals and

whether it is an application file (this parameter is not used for most cases). The comma is used as a separator. The number of data lines is specified by the “number of intervals” part of the control line. Every data line includes 3 items: low bound of an interval, high bound, and probability for this interval. The following file gives an example IDF file:

```
4,0
0,0.25,0.2
0.25,0.50,0.3
0.5,0.75,0.25
0.75,1.0,0.25
```

This file describes the distribution of a variable consisting of 4 bars from 0 to 1. A sample file is a special .IDF. In this type of file, the low bound of every interval is set to equal the high bound. It just is used to say how much the probability at this point is.

Two different ways are used to input data in Statool. One is to directly load the distribution of a random variable from a file. The other is to edit the distribution of a variable. To use the editor to input the distribution, Xie (1998) gave a detailed description.

There are two ways to output the result of an operation. One is to draw the graph of the variable. It is convenient for seeing the result immediately. The other way is to save data as an IDF or SMP file. These two files are discussed in the previous paragraph. The user can permanently keep data in and use these files in the future.

There are 2 types of graph to show the data: probability bars for PDF format and probability bound curves for CDF format. From the probability bars of PDF format, the user can see the histogram of random variable. Probability bound curves show envelopes bounding the space of possible cumulative probability functions of a random variable.

### **4.3 The user interface**

"A user interface is an interface that enables information to be passed between a human user and hardware or software components of a computer system." [IEEE, Std 610.12-1990]. Based on the Microsoft platform system, windows are used to implement the user interface. Generally, the user interface consists of various windows. They can be moved on the screen, overlap each other and minimized into icons on the task bar of MS operating system.

In general, a program running on an MS platform includes the primary windows and the secondary windows. The primary windows handle major interactions with the user, and often contain an arbitrary number of objects. Secondary windows are used to support the interactions with primary windows by providing details about the objects of the primary window and operations on those objects.

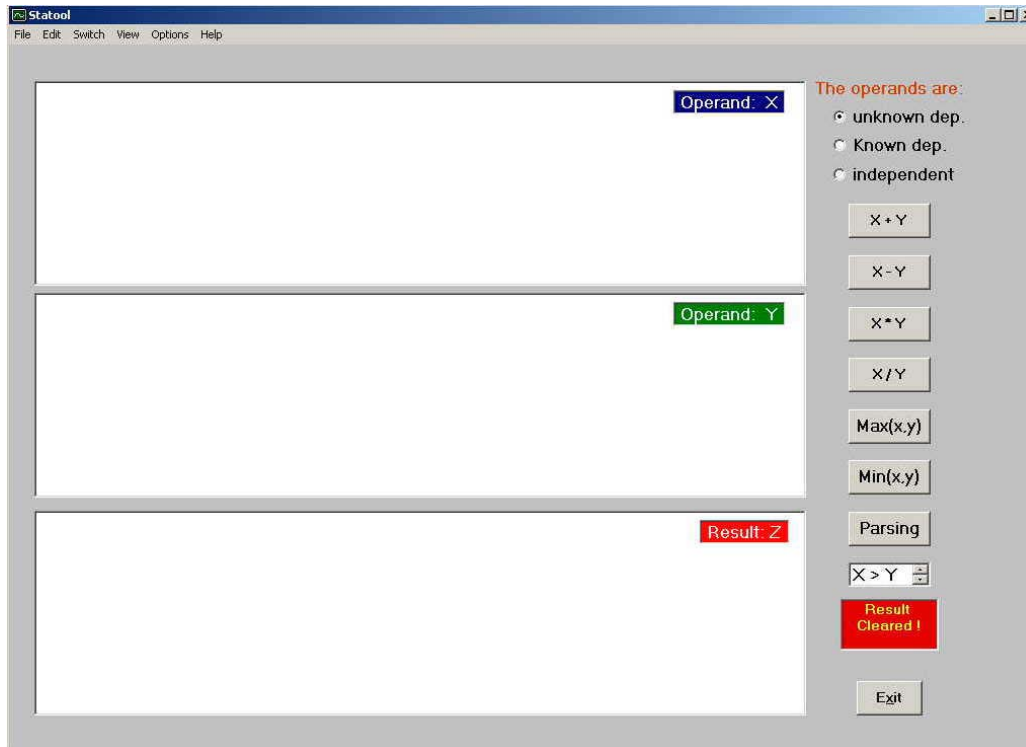
Statool is comprised of these two types of windows. There are three primary windows and there are secondary windows for each of the primary windows. The three primary windows are the operation window, the data editor window, and the data view window. The operation window is the main primary window. The other two primary windows can be accessed from this main primary window. To decrease interaction overhead, window navigation paths are restricted to three levels. Otherwise the user will be likely to get lost in the system. The operation window is opened when the user starts the program and is always open as long as the program is running.

#### **4.3.1 Main primary window: the operation window**

The operation window is the place to perform the binary operations. It also includes some associated functions such as file operations and setting options. The user can use this window to perform the following functions:

- Data maintenance ( including load, save, view, and edit data)
- Data operation ( choosing the operation type)
- Options for data operation
- Help and additional information about this software
- Control software (including program termination and options to run this software)

Based on these functions requirement, this figure gives a prototype interface for it.



**Figure 4.2. Operation window.**

This window consists of five parts: menu bar, data display panels, option area for correlation, operation area, and exit button. The menu bar is used to contain all functionalities' entries. Data display panels include 3 panels. Two panels are used to store operand X and Y, and a third one to show the operation result Z. The option area is used to choose the correlation for operands X and Y. Currently, 3 types of correlation are supported for this software. They are independent, unknown, and known correlation. the operation area contains all the operations. The user can choose the specific operation to calculate. The exit button is a convenient way to exit the program although you can also do this by selecting the exit option on the menu bar.

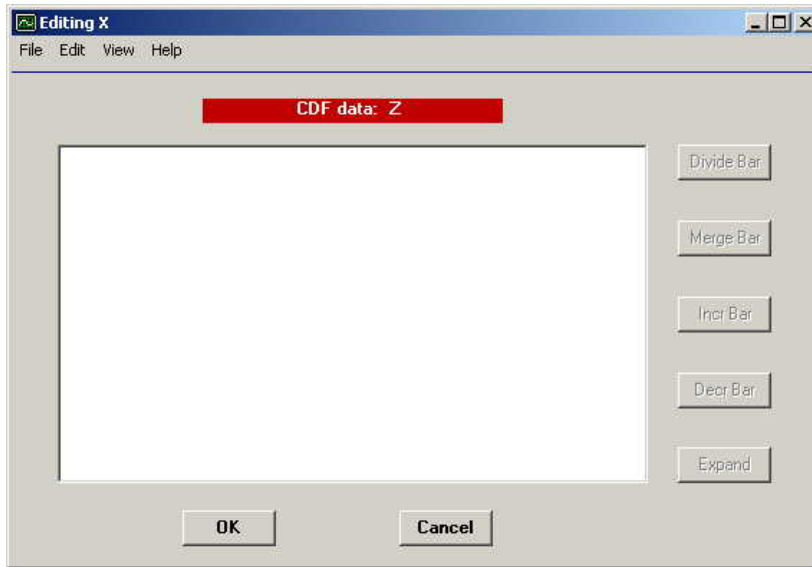
The operation window includes 5 property windows, which are the secondary windows. They are the display mode window, display color setting window, about window, correlation value setting window, and expression editor window. The display mode window is used to choose the data display mode: PDF or CDF. The display color setting window is used to choose the colors for the 3 display panels. The about window lists information about



this software. The correlation value setting window enables the user to input information about correlation when the user selects the known correlation radio button.

#### 4.3.2 Other primary windows: the data editor and the view windows

The data editor window is used to edit the input data, which is shown in the following figure:

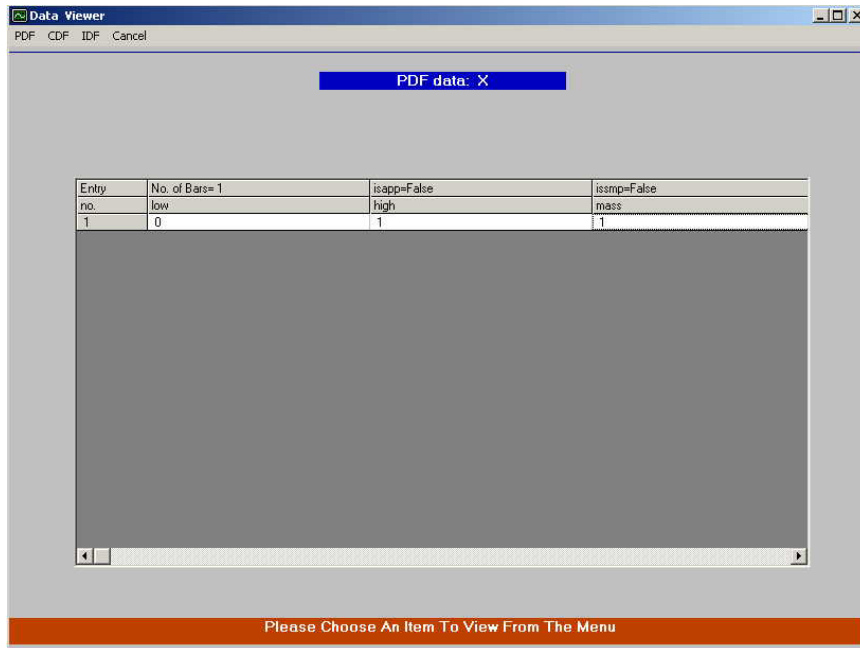


**Figure 4.3. Data editor.**

This window includes a menu bar, a display panel for data, operations for editing data, and control buttons for the window.

The data editor window includes 4 property windows: Bar number, Value range, Value for single bar and About. The bar number window is used to set the total number of bars. The value range window is used to set the range for the operand. The value for single bar window controls the probability and the width for this bar. The about window is the same as the operation window.

The data view window is used to show the exact values for data. It is shown in the following figure:



**Figure 4.4. Data view.**

This window consists of 2 parts: menu bar and display table for data. The user can choose the expected data format to show in the table. The data view window does not include any property windows.

### 4.3.3 Windows management

All windows of Statool are compatible with the standard of MS windows. Every window includes the standard functions: move, size, min, max, restore and close. The following visual dimensions will be considered: position, size, shape and color.

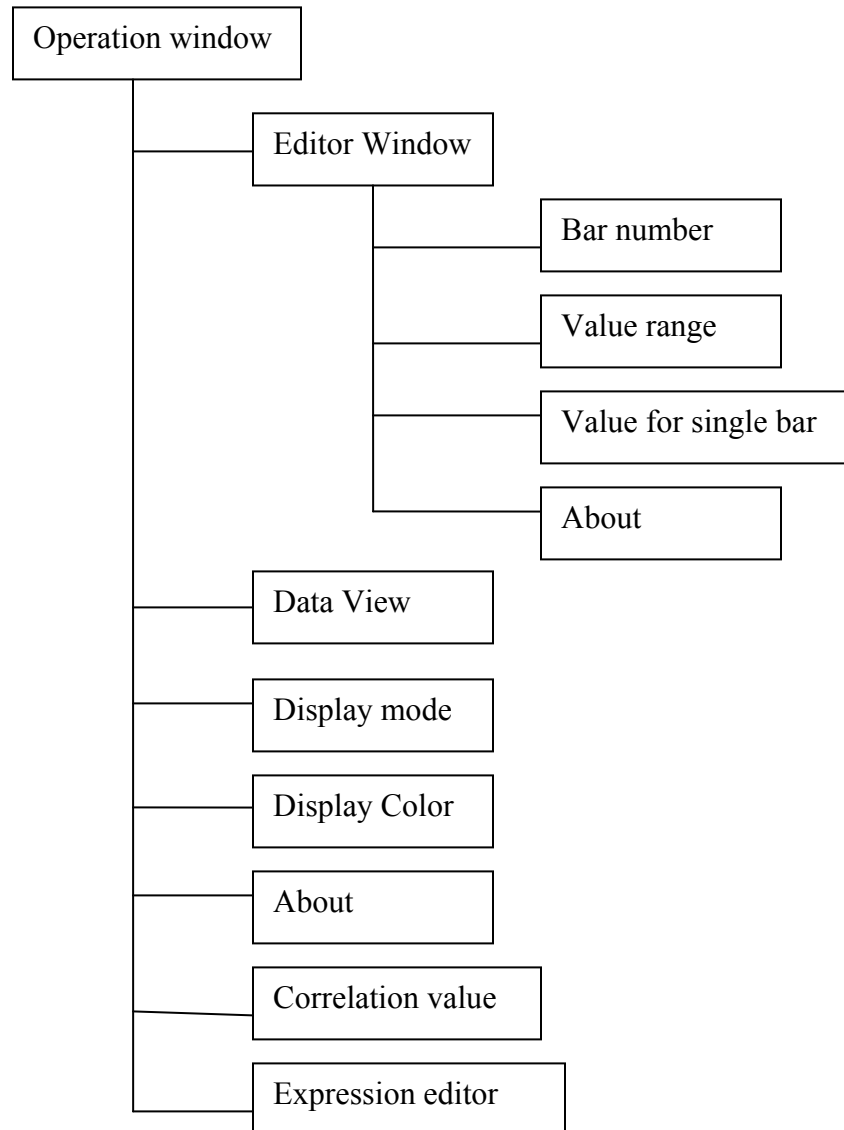
Position: every window will be placed in the central of the current screen.

Size: the main primary window will not exceed 640x480 pixels to make sure it can be shown on any monitors. Other primary windows will not exceed the size of the main primary window. Property windows can't exceed the size of their primary windows.

Shape: set different shapes for different objects.

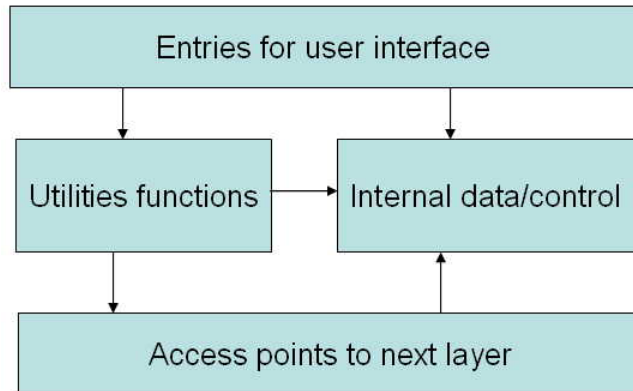
Color: keep the colors consistent for the same parts of different windows, such as background and foreground color.

The length of the windows' navigation path is three. The following figure shows the windows' navigation path:



#### **4.4 The logical layer**

This layer lies between the user interface and the computing layer. It contains access points from the user interface, all logical controls and internal data, utility functions and access points to call the next layer. This layer just provides the subsystems to service the user interface and doesn't perform any actions related to the "real" functions: uncertainty operations, which are the real work of this software. Therefore, it makes sense to call it the business-specific layer. The figure shows the structure of this layer:



**Figure 4.5. Logical layer.**

The major function of the logical layer is to map user actions to the logical functions' view of the software. For example, the user might click to load a file from storage into memory in the logical view. This layer will do the real work in response to an action happening in the user interface. This part mainly includes the response functions for the menu bar and response functions for the radio buttons displayed in the user interface.

This layer will also provide the many utility functions and classes and maintenance of internal data structures to keep the important state information.

Finally, this layer provides a uniform interface to the computing layer. All the computing sensitive work is put into the computing layer. To use and manage these modules efficiently, the standard interface is necessary. It also increases the reusability of these modules.

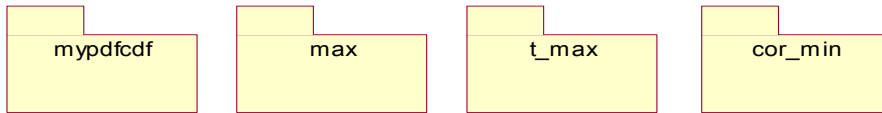
In summary, this layer is used to formulate the user problem to fit the developed uncertainty algorithm. Three main functions are provided in this layer: logical functions to respond to the user actions, maintenance state information to control uncertainty operation, and efficient access points to call the computing layer.

## **4.5 The computing layer**

This layer is the computing subsystem. It implements almost all the algorithms of the uncertainty operations. It is obvious that this layer is computing sensitive and will consume a lot of computing and memory resources.

This layer was developed separately from the other two layers since it does not really depend on the other two layers. A benefit is that this layer can be extended to run on other computing resources to decrease its dependency on computing resources at the client side. Based on the MS platform, a dynamically linked library (DLL) was used. Calling a DLL is language independent. Any development languages can call this type of library by following the calling conventions.

This layer includes mainly 4 packages: converting data, general simplex method, transportation simplex method, and extended simplex method. These 4 packages are independent of each other. There is no calling relationship among them. Every package just finishes the specific function using the corresponding algorithm.



**Figure 4.6. Package view for the computing layer.**

## **5 Component design and implementation**

This chapter lists detailed information about components belonging to different layers. At the same time, implementation issues also are included with components design.

### **5.1 Overview**

Components design is closely related to the development language. For the user interface and the logical layer, MS Visual Basic is used since it is a very popular rapid development tool. Each window maps to a form. Response actions map to a function or routine of this form. So these two parts are integrated together. This is a convenient way for implementing although separated from the logical view. Internal data structures and utility functions and classes will be defined in module files of visual basic. For the computing layer, Visual C++ is used since DLLs developed in it provide acceptable computing speed.

### **5.2 Operation and properties windows**

The operation window is the primary window for this program. So it is the main form (also the starter form) for the visual basic implementation. The compiled program, “Statool” will start from this form. This window also includes 5 properties windows: display mode, display color, about, correlation value, and expression editor.

#### **5.2.1 Operation window**

This form is called frmMain in the project form view. The following figure indicates this form:

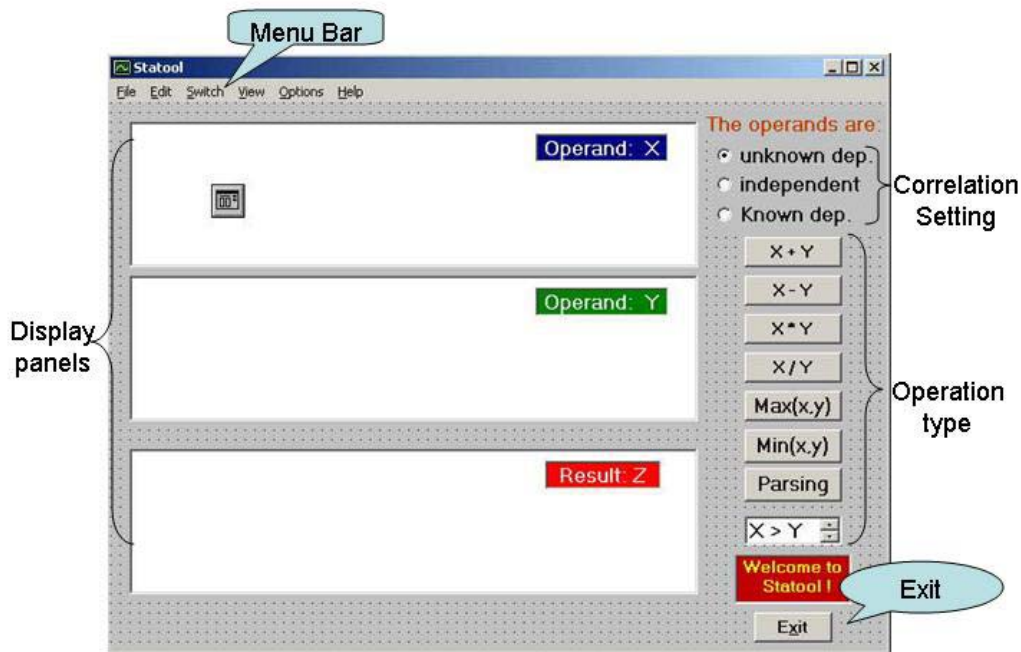


Figure 5.1. Operation window design.

### 5.2.1.1 Menu bar

There are six items in the menu bar. They are File, Edit, Switch, View, Options, and Help.

**File:** manage files and control program running. This figure shows its submenu:

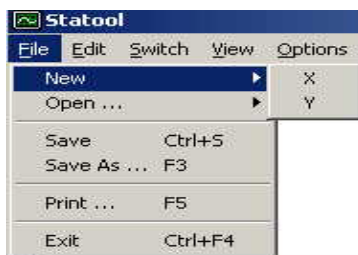


Figure 5.2. File menu.

It includes 6 submenus: New, Open, Save, Save As, Print and Exit.

New: create a new IDF file for operand X or Y. Response subroutine is `mnuNewX_Click()` for new X and `mnuNewY_Click()` for new Y in form: `frmMain`.

**Open:** open the existing file for operand X or Y. Response subroutine is `mnuOpenX_Click()` for open X and `mnuOpenY_Click()` for open Y.

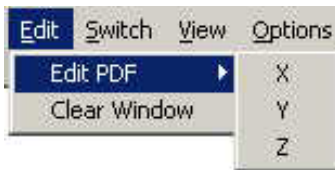
**Save:** save the current result Z. Response subroutine is `mnuSave_Click()`.

**Save As:** save the current result Z as a specific file. Response subroutine is `mnuSaveAs_Click()`.

**Print:** print the current operation window. Response subroutine is `mnuPrint_Click()`.

**Exit:** terminate the current program. Response subroutine is `mnuExit_Click()`.

**Edit:** edit the specific operand. This figure shows its submenu:

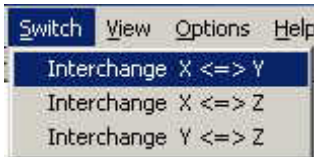


**Figure 5.3. Edit menu.**

**Edit PDF:** activate the editor window to edit the PDF file chosen from X, Y and Z. Response subroutine is `mnuEditX_Click()` for operand X, `mnuEditY_Click()` for operand Y, and `mnuEditZ_Click()` for operand Z.

**Clear Window:** clear the specific display panel. Response subroutine is `mnuClearwin_Click()`.

**Switch:** switch the files between 2 the 3 main window panels. It provides a convenient way to exchange displays among the three display panels. This figure shows its submenu:



**Figure 5.4. Switch menu.**

**Interchange X  $\Leftrightarrow$  Y:** switch operands X and Y. Response subroutine is `mnuInterXY_Click()`.

**Interchange X  $\Leftrightarrow$  Z:** switch operands X and Z. Response subroutine is `mnuInterXZ_Click()`.

**Interchange Y  $\Leftrightarrow$  Z:** switch operands Y and Z. Response subroutine is `mnuInterYZ_Click()`.



**View:** provide the entries to set the display mode for panels and activate the data view window. This figure shows its submenu:

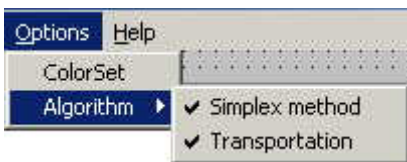


**Figure 5.5. View menu.**

**Display Mode:** activate the property window to set the display mode for operands. Response subroutine is `mnuDisplayMode_Click`.

**Data:** activate data view primary window. Response subroutine is `mnuData_Click`.

**Options:** set options controlling the program and its operation. This figure shows its submenu:

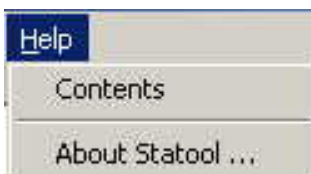


**Figure 5.6. Options menu.**

**ColorSet:** activate the property window to set the display color for panels. Response subroutine is `mnuColorSet_Click`.

**Algorithm:** set the preferred algorithm to handle the linear programming problems. There are two choices: Simplex method and Transportation. Response subroutine is `mnuSimple_Click` for Simplex method, and `mnuTransportation_Click` for Transportation.

**Help:** display help information and about information for this software. This figure shows its submenu:



**Figure 5.7. Help menu.**

Contents: activate the help information. Response subroutine is `mnuContent_Click`.

About Statool: activate the property window to show the about information. Response subroutine is `mnuAbout_Click`.

### 5.2.1.2 Display panels

There are three display panels to contain operands X and Y, and result Z. These display panels use PictureBox (one type of Visual Basic visual component). Their names are `picX`, `picY`, and `picZ`. They are used to show the graphical representation of PDF bars or CDF bound curves according to the display mode for X, Y and Z.

Every panel provides a popup menu to support mouse functions. When the right button of the mouse is clicked, a popup menu will be shown. Response subroutine is `picX_Click` for operand X, `picY_Click` for Y, and `picZ_Click` for Z. The following figures show 3 popup menus for 3 display panels.



Figure 5.8. Popup menu for operand X.



Figure 5.9. Popup menu for operand Y.

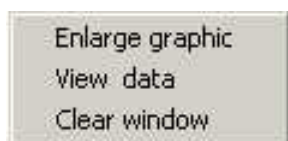


Figure 5.10. Popup menu for result Z.

These popup menus provide the same functionalities as the menu bar. But its implementation is different from that of the menu bar. Operands X and Y use the same popup

item object, whose name is Popupitem. Response subroutine for choosing popup menu is Popitem\_Click. Result Z uses its own popup item object, called PopZitem. Response subroutine is PopZitem\_Click to handle menu commands.

### 5.2.1.3 Correlation setting

This part consists of 3 option buttons (a type of visual component in VB). They correspond to 3 types of relationships between X and Y. They are unknown dependence, known dependence and independence. These 3 option buttons are exclusive since only one situation is true for the current operands X and Y. This figure shows this part.

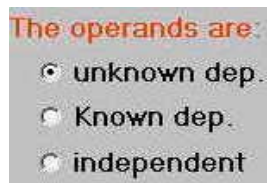


Figure 5.11. Correlation setting.

Response subroutine is optDep\_Click for unknown dep., optInd\_Click for independent, and OptCorrelation\_Click for known dep.

### 5.2.1.4 Operation type

This part consists of seven command buttons and one listbox (both are the basic visual components for visual Basic), whose names are cmdOp1 to cmdOp8. By choosing these buttons and listbox, the user can choose the different operations for operands X and Y. This figure shows this part.



**Figure 5.12. Operation types.**

Button “X+Y” performs the addition operation between X and Y. Response subroutine is cmdOp1\_Click. Button “X-Y” performs the subtraction operation between X and Y. Response subroutine is cmdOp2\_Click. Button “X\*Y” performs the multiplication operation between X and Y. Response subroutine is cmdOp3\_Click. Button “X/Y” performs the division operation between X and Y. Response subroutine is cmdOp4\_Click. Button “max(x,y)” finds maximum value distribution envelopes from X and Y. Response subroutine is cmdOp5\_Click. Button “min(x,y)” finds minimum value distribution from X and Y. Response subroutine is cmdOp6\_Click. Button “Parsing” activates the property window to input expression consisting of X and Y. Then it evaluates this expression using X and Y. Response subroutine is cmdOp7\_Click. Listbox performs relational operations between X and Y, which include 4 types of relationship: greater than, not less than, less than, and not greater than. Response subroutine is cmdOp8\_Click.

#### **5.2.1.5 Exit button**

This button provides a convenient way to terminate this program. If the user wants to exit this program, by clicking this button, this program will exit. Response subroutine is cmdExit\_Click.

## 5.2.2 Properties windows of the operation window

For the operation window, there are 5 properties windows. Their windows are used to set the parameters and control information for the operation window. They are Display mode, Display color, About, Correlation Value, and Expression editor.

### 5.2.2.1 Display mode

This property window is used to set the display mode for operands X/Y and result Z. There are 2 types of display modes: PDF bars and CDF envelope curves. Response form is called frmDisplay. This figure shows it:

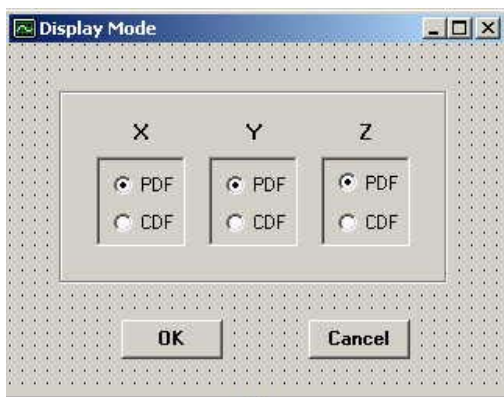
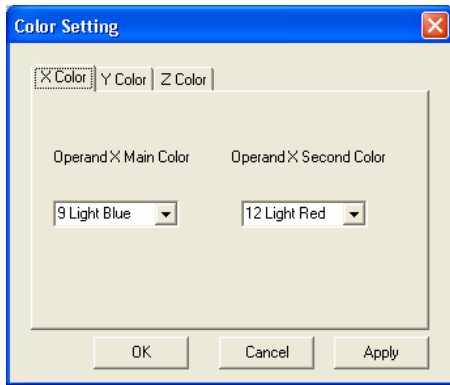


Figure 5.13. Display mode window.

Choosing the different display mode for X, Y and Z will change the internal control variables `da_pdfmode`, `db_pdfmode`, and `db_pdfmode`. When values for these variables are 1, PDF mode is chosen. Otherwise CDF mode is chosen.

### 5.2.2.2 Display Color

This window controls the color for three display panels. Response form is `frmColorSet`. The following figure shows it.



**Figure 5.14. Display color window.**

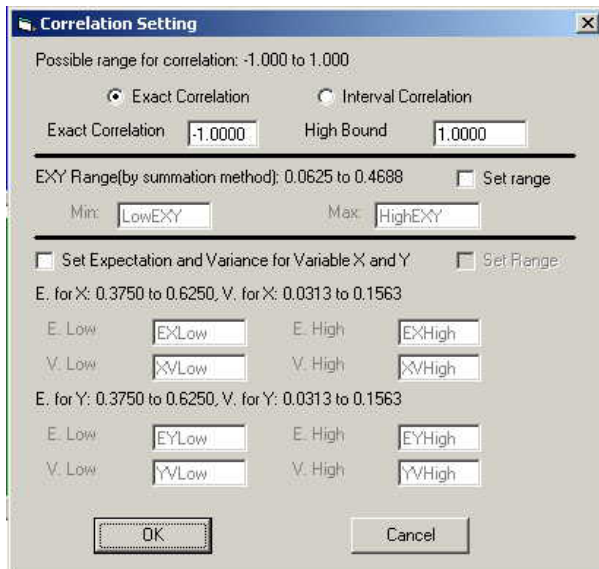
This window controls the color variables for X, Y and Z. These variables are XMainColor, XSecondColor, YMainColor, YSecondColor, ZMainColor, and ZSecondColor.

### 5.2.2.3 About Window

This window displays the copyright information for this software.

### 5.2.2.4 Correlation Value Window

This window is used to set any known information about correlation. Response form is DlgCorSet. When user choose the option “Known dep..” from the operation window, this window will be activated. Through this window, the user can input any known correlation information that may narrow the CDF envelope curves. This figure shows this window:



**Figure 5.15. Correlation setting window.**

From this window, there are 3 ways to input information about operand X and Y: known correlation range or exact value, known expectation range for XY, and known expectation and variance for X and Y. .

### 5.2.2.5 Expression editor window

User can input expressions consisting of two variables X and Y. Response form is frmExpression. When user clicks the “Parsing” button on the main window, this window will be activated so that the user can input and edit the expression desired. This figure shows this window:

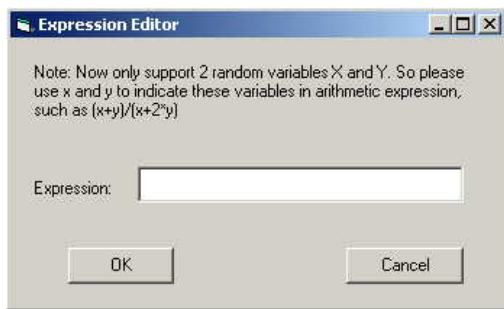


Figure 5.16. Expression editor window.

The expression input by the user will be evaluated using intervals in the discretizations of X and Y. This form will call the utility functions in the modules file, named MathParserRoutines. All the functions related to expression parsing are saved in this modules file.

## 5.3 Other primary windows

The Data editor window provides the data editing function for the user. The user can use this editor to create and modify operands X and Y. Response form is frmHistEdit.

The Data View window is used to display the data of X, Y and Z when user needs numerical output instead of graphs. Response form is frmViewer.

## 5.4 Lower 2 levels of logical layer

From the structure of the logical layer, the lower 2 levels consist of utility functions, internal data structures, and access points to the next layer.

### 5.4.1 Utility functions and internal data structure

This level consists of two module files: Module1 and MathParserRoutines. All the internal data structures are defined in Module1. This module also includes the basic utility functions, such as load and save data. MathParserRoutines just provides the support functions for the expression parser.

All the data structures are defined as public in the module file. This way, all other forms and subroutines can access them directly. There are 2 main types of internal data structures: parameters to control uncertainty operation and program, and data buffers to store data of X, Y, and Z.

Utility functions are related to file management and to drawing graphs in the three display panels. There also is a very important utility function for synchronizing data among the different data types for the same random variable.

As the previous described, there are four types of data used by the software. Generally, only the .IDF or .SMP files are used to input or output data. So there is a need to keep other type data consistent with it. The different types of data describe the same variable from different points of view. But they still say the same thing, so they can be transformed from one to another. To execute a transformation, there are different requirements for the different types of random variables. The transformation can be computationally expensive. It is not suitable to do the transformation directly in this function, so it is put into the computing layer. This utility function is defined as following:

#### **HavePdfCdf (distribution-type as integer)**

This subroutine has only one parameter specifying the type of distribution. The parameter value determines if the data structure under transformation is an operand variable or result variable.

### 5.4.2 Access points to the computing layer

There are 4 packages in the computing layer. Every package provides the entry points for the layer above.

For the standard data package, there are 2 access points as follows:

```
Public Declare Sub GetPdfCdf Lib "mypdfcdf.dll" Alias "_mypdfcdf@36" (idfno As Long,
```



isapp As Long, idfdata As Single, pdfno As Long, pdfdata As Single, lobound As Single, upbound As Single, cdfno As Long, cdfdata As Single)

Public Declare Sub GetPdfCdf\_1 Lib "mypdfcdf\_1.dll" Alias "\_mypdfcdf@40" (idfno As Long, isapp As Long, idfdata As Single, pdfno As Long, pdfdata As Single, lobound As Single, upbound As Single, cdfno As Long, cdfdata0 As Single, cdfdata1 As Single)

For the legacy simplex method, there is 1 access point as follows:

Public Declare Function Max Lib "Max.dll" Alias "\_Max@20" (objective As Double, X As Double, Y As Double, ByVal m As Long, ByVal n As Long) As Double

For the transportation simplex method, there is 1 access point as follows:

Public Declare Function T\_Max Lib "T\_Max.dll" Alias "\_T\_Max@20" (objective As Double, X As Double, Y As Double, ByVal m As Long, ByVal n As Long) As Double

For the extended simplex method, there are 4 access points as follows:

Public Declare Function Cor\_Min Lib "Cor\_Min.dll" Alias "\_Cor\_Min@28" (objective As Double, X As Single, Y As Single, ByVal m As Long, ByVal n As Long, ByVal LowB As Single, ByVal HighB As Single) As Double

Public Declare Function Cor\_Bound Lib "Cor\_Min.dll" Alias "\_Cor\_Bound@40" (X As Single, Y As Single, ByVal m As Long, ByVal n As Long, LowEXY As Single, HighEXY As Single, LowB As Single, HighB As Single, XValue As Single, YValue As Single) As Integer

Public Declare Function Cor\_Min\_Exy Lib "Cor\_Min.dll" Alias "\_Cor\_Min\_Exy@40" (objective As Double, X As Single, Y As Single, ByVal m As Long, ByVal n As Long, ByVal LowB As Single, ByVal HighB As Single, ByVal maner As Long, EXYLow As Single, EXYHigh As Single) As Double

```
Public Declare Function Cor_Min_Exy_S Lib "Cor_Min.dll" Alias "_Cor_Min_Exy_S@44"
(objective As Double, X As Single, Y As Single, ByVal m As Long, ByVal n As Long,
ByVal LowB As Single, ByVal HighB As Single, ByVal maner As Long, EXYLow As
Single, EXYHigh As Single, ByVal callcount As Long) As Double
```

## 5.5 The computing layer

All the algorithms are implemented in this part. They are computing intensive and time-consuming. This part consists of 4 packages. They are Standard Data, general simplex method, Transportation simplex method, and extended solution using correlation as a constraint.

### 5.5.1 Converting data

This package provides the implementation for converting data from one format to another. It is provided using a dynamically linked library (DLL). This DLL is named “mypdfcdf.dll”. This DLL includes just one method for converting data. This method is called “mypdfcdf”. Its exact definition using C++ is listed as following:

```
#ifdef __cplusplus
extern "C" __declspec( dllexport ) void __stdcall mypdfcdf( int * idfno, int *
isapp, float idfdata[][3], int * pdfno, float pdfdata[][3], float * lobound, float *
upbound, int cdfno[], float cdfdata[][1025][3] )
#endif
```

#### parameters:

int \*idfno: the address of the number of the idf’s interval, if any  
int \*isapp: the address of whether it is application data  
float idfdata[][3]: the 2 dimension array of idf data, if any. Each row includes 3 columns: low bound of interval, high bound, and probability for this interval.  
int \*pdfno: the address of the number of the pdf’s interval, if any.  
float pdfdata[][3]: the 2 dimension array of pdf data, if any. Formation is the same as that of idf  
float \*lobound: the address of the low bound  
float \*upbound: the address of the up bound

int cdfno[]: the array for the number of cdf bounds

float cdfdata[][1025][3]: the 3 dimension array for cdf data

To support big size data, this package also provides an extended DLL, called mypdfcdf\_1.dll. This DLL does the same work as the original DLL, but the parameters of the function are changed to support big size data. Its definition is as follows:

```
extern "C" __declspec( dllexport ) void __stdcall mypdfcdf( int * idfno, int *  
isapp, float idfdata[][3], int * pdfno, float pdfdata[][3], float * lobound, float *  
upbound, int cdfno[], float cdfdata0[][3], float cdfdata1[][3])
```

Comparing it with the previous definition, only the last two parameters are different. In the extended version, parameter cdfdata is split into 2 parameters. This will remove the constraint the previous definition imposes of defining the size of array.

Although this function is defined in a “C++” file, it still requires the standard “C” string in the declaration since this interface will be put into a DLL and called by another language. So this interface has to be consistent with the requirement to use a “C” standard in the declaration. On the Microsoft platform, there are three calling conventions: \_\_cdecl, \_\_fastcall, and \_\_stdcall. The different calling conventions define the different stack push methods. We use only \_\_stdcall. Microsoft developer network (MSDN) gives detailed information about how to make DLLs to be used by another language.

In this DLL, PDF information is often approximate and is derived from IDF format for display purposes. CDF format has two sets of data: minimum probability and maximum probability for the specified point on the horizontal axis. At most 64 bars are supported in the DLL for PDF format. Probability will be assigned to these bars according to IDF data, so the result for this transformation is approximate except when the IDF has no overlaps in it. The following pseudo code describes how it works.

```
/* First get PDF from IDF */
```

```
If IDF no overlap then
```

```
    Set lowbound as lowbound of first bar of IDF
```

```
    Set highbound as highbound of last bar of IDF
```

```
    Set PDF same as IDF
```

Else

Number of bars <- max(number of bar of IDF, PDFMAX)

Set Lowbound as lowbound of first bar of IDF

Set Highbound as highbound of last bar of IDF

Average Width of PDF bar <- ( Highbound – Lowbound) / Number of bars

Set low/high bound for every bar of PDF according to Lowbound and bar width

Assign the probabilities of PDF bars from IDF bars

End if

/\* Then get CDFs from IDF, CDF1 contains data of high bound curve, CDF2 contains data of low bound curve\*/

Number of bars of CDFs <- number of IDF +1

Lowbound of CDF1<- lowbound of each bar of IDF

Highbound of CDF2<-highbound of each bar of IDF

Order bars of CDF1 according to the lowbound of each bar

Order bars of CDF2 according to the highbound of each bar

Merge the same bars of CDF1 according to lowbound of each bar

Merge the same bars of CDF2 according to highbound of each bar

Set the highbound of each bar of CDF1

Set the lowbound of each bar of CDF2

Assign the probability to each bar of CDF1 based on IDF

Assign the probability to each bar of CDF2 based on IDF

Get the cumulative probability of each bar of CDF1

Get the cumulative probability of each bar of CDF2

### 5.5.2 General (legacy) simplex method

This package was inherited from the previous version. It uses the standard simplex method to do linear programming. It was provided as a dynamically linked library. Its name is max.dll. Its interface is defined as follows:

```
extern "C" __declspec( dllexport ) double __stdcall Max( double* objective, double* x,
double* y, int m, int n);
```

Xie (1998) gave detailed information about this interface.

### 5.5.3 Transportation simplex method

The transportation simplex method was implemented in this package. This package also is provided as a dynamically linked library (DLL). The algorithm was discussed in the previous chapter. In this part, we just focus on some details about design and implementation.

This component is called `t_max.dll`. The function is named `T_Max` (please note capitals in name). The full definition is listed as follows:

```
#ifdef __cplusplus
    extern "C" __declspec(dllexport) double __stdcall T_Max(double *objective, double
    *x, double *y, int m, int n)
#endif
```

Parameters:

double \*objective: array for cost coefficient for every cell

double \*x: marginal distribution for operand x

double \*y: marginal distribution for operand y

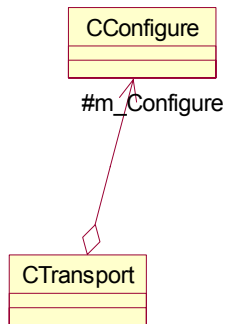
int m: the number of bars for operand x

int n: the number of bars for operand y

This function's definition is kept the same as the original interface called `max`. Xie (1998) listed detailed information about its parameters. This function will return the maximum sum of probabilities for cells specified by the parameter named `objective`. This DLL is stateless. To call this function, parameters must be passed for every call although they are the same except for the parameter called `objective`.

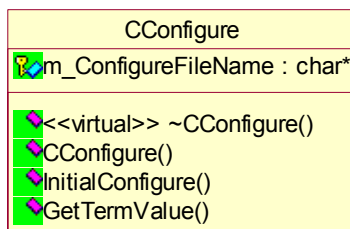
The transportation simplex method is implemented in a class called `CTransport`. There is another utility class, called `CConfigure` in this package. Figure 4.4 shows the

relationship between these classes.



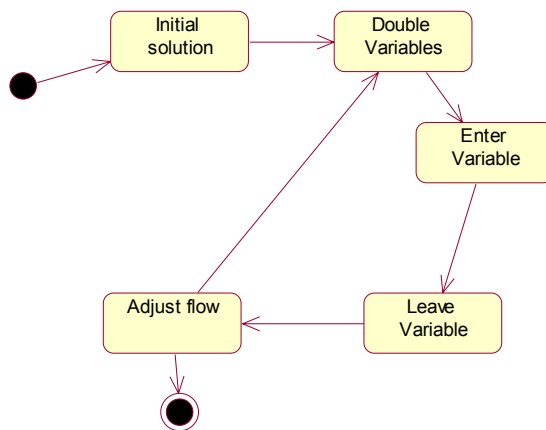
**Figure 5.17. Class relationship for transportation simplex method.**

Class CConfigure will read parameters from a configuration file to control the transportation simplex method. Currently, there is only one parameter to control which initialization method is used to get the initial feasible solution: Northwest corner method and Russell approximation method.



**Figure 5.18. Class CConfigure.**

Class CTransport implements the transportation simplex method. This figure shows the flow of control:



**Figure 5.19. Flow of control for transportation Simplex implementation.**

Handling degeneracy is a key part of this algorithm. Generally speaking, degeneracy means the number of basic variables is less than  $M+N-1$  for an  $M \times N$  matrix. The reason for it is a little complex. The number of basic variable is determined by the procedure for getting an initial feasible solution. This number can't be changed during subsequent iterations. So the initial feasible solution is very important. Different initialization methods will produce different initial feasible solutions. For example, the Northwest Corner approach is easy, simple, and provides enough basic variables although maybe the value of some basic variables is zero. Russell's approximation method provides a good initial solution, which is very close to the best solution, so subsequent computation will be lessened. But it may fail to provide an initial feasible solution with enough basic variables. Two initial procedures were implemented in this class: the Northwest corner method and Russell's approximation method. The Northwest corner method is the default method to get the initial feasible solution. The user can choose Russell's method as the initialization method through changing a parameter in the configuration file. This is just a way provided for the advanced user who knows what they are doing.

#### **5.5.4 Incorporating correlation as a constraint**

This package is in `cor_min.dll`. Four functions are provided in this package. They are `Cor_Min`, `Cor_Min_Exy`, `Cor_Min_Exy_S`, and `Cor_Bound`. The first 3 are used to operate on random variables when correlation is set. Among these three, each subsequent one has a

super set of functions compared with the previous one. The last one is used to get the estimated range of possible correlations for the two operands.

The first three functions are defined as the follows:

```
extern "C" __declspec( dllexport ) double __stdcall Cor_Min(double *pCost, float pX[][3],
float pY[][3], int mx, int ny, float LowBound, float HighBound);
```

Parameters:

- double \*pCost: array for cost coefficient for every cell
- double \*pX: 2 dimension array to contain every bar of operand x
- double \*pY: 2 dimension array to contain every bar of operand y
- int m: the number of bars of operand x
- int n: the number of bars of operand y
- float LowBound: lowbound of correlation
- float HighBound: highbound of correlation

```
extern "C" __declspec( dllexport ) double __stdcall Cor_Min_Exy(double *pCost, float
pX[][3], float pY[][3], int mx, int ny, float LowBound, float HighBound, int
EXYManner,float *EXYLow,float *EXYHigh);
```

Parameters:

- double \*pCost: array for cost coefficient for every cell
- double \*pX: 2 dimension array to contain every bar of operand x
- double \*pY: 2 dimension array to contain every bar of operand y
- int m: the number of bars of operand x
- int n: the number of bars of operand y
- float LowBound: lowbound of correlation
- float HighBound: highbound of correlation
- int EXYManner: indicate whether value of EXY set by the user is used
- float \*EXYLow: address of the lowbound of EXY
- float \*EXYHigh: address of the highbound of EXY



```
extern "C" __declspec( dllexport ) double __stdcall Cor_Min_Exy_S(double *pCost, float
pX[][3], float pY[][3], int mx, int ny, float LowBound, float HighBound, int
EXYManner,float *EXYLow,float *EXYHigh,long CallCount);
```

Parameters:

double \*pCost: array for cost coefficient for every cell  
double \*pX: 2 dimension array to contain every bar of operand x  
double \*pY: 2 dimension array to contain every bar of operand y  
int m: the number of bars of operand x  
int n: the number of bars of operand y  
float LowBound: lowbound of correlation  
float HighBound: highbound of correlation  
int EXYManner: indicate whether value of EXY set by the user is used  
float \*EXYLow: address of the lowbound of EXY  
float \*EXYHigh: address of the highbound of EXY  
long CallCount: counter to count the time to call this interface

From the previous definitions, it is evident that more parameters are required for the later functions. So the later functions have more functionality than the earlier ones.

The function to estimate the correlation range is not classified to other three. But they all have some similarities and use some same utilities. So they are put together into a package (DLL). Here is the interface of that 4<sup>th</sup> function.

```
extern "C" __declspec( dllexport ) int __stdcall Cor_Bound(float pX[][3], float pY[][3], int
mx, int ny, float *LowK, float *HighK, float *LowCor, float *HighCor, float *XValue, float
*YValue);
```

Parameters:

double \*pX: 2 dimension array to contain every bar of operand x  
double \*pY: 2 dimension array to contain every bar of operand y  
int m: the number of bars of operand x  
int n: the number of bars of operand y

float \*LowK: address to store the lowbound of possible EXY  
float \*HighK: address to store the highbound of possible EXY  
float \*LowCor: address to store the lowbound of possible correlation  
float \*HighCor: address to store the highbound of possible correlation  
float \*XValue: address to store the expectation and variance of operand X  
float \*YValue: address to store the expectation and variance of operand Y

#### 5.5.4.1 Class structure

This package provides 2 main functions: computing the possible correlation range based on marginal distributions of operands X and Y; and getting CDF bounds' values according to the correlation setting. To perform these functions, there are 8 classes defined in this package. They are COptimalMin, CConfig, CSimplex, CVariance, CMaxmin, CBoundCorrelation, CMinCorrelation, and CMinCorrelationExy.

COptimalMin: getting the minimum value of the specified non-linear functions on the specified variables range.

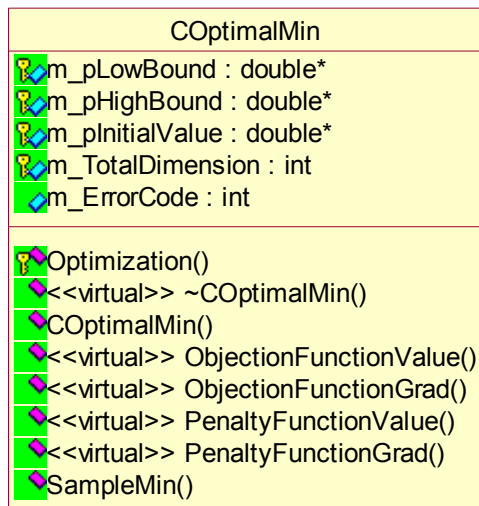


Figure 5.20. Class: COptimalMin.

CConfig: reading the configuration information from the configuration file: Statool.cfg.






CConfigure
 m_ConfigureFileName : char*
 <<virtual>> ~CConfigure()  CConfigure()  InitialConfigure()  GetTermValue()

Figure 5.21. Class: CConfigure.

CVariance: getting the min/max value of variance of random variable.

















CVariance
 m_pCoefficient : double*
 m_Expectation : double
 m_Variance : double
 CalculateMean()  CalculateVariance()  <<virtual>> ~CVariance()  CVariance()  ROFValue()  ROFGrad()  OFValue()  OFGrad()  PFValue()  PFGrad()  SetParameter()  GetMin()  GetMax()

Figure 5.22. Class: CVariance.

CSimplex: solving the linear programming problem using improved simplex method.





















































CSimplex	
	MAX_LOOPNUMBER : long
	LEAVE_EPS : double
	INVERSE_EPS : double
	EPS : double
	m_pDebugFile : FILE*
	m_Version : short
	m_Status : short
	m_pBuffer2 : double*
	m_pBuffer1 : double*
	m_PreviousLeaveOrder : short
	m_PreviousEnterOrder : short
	m_PreviousLeaveVariable : short
	m_PreviousEnterVariable : short
	m_LeaveOrder : short
	m_EnterOrder : short
	m_LeaveVariable : short
	m_EnterVariable : short
	m_NumberofBasic : short
	m_NumberofNonbasic : short
	m_BigConstant : double
	m_plInverseB : double**
	m_pNonbasic : short*
	m_pBasic : short*
	m_pCost1 : double*
	m_pWi : double*
	m_pWi1 : double*
	m_NumberofZeroCoefficient : int
	m_ErrorCode : int
	m_Min : double
	m_pb : double*
	m_pCost : double*
	m_pMatrixA : double**
	m_NumberofVariable : short
	m_NumberofConstraint : short
	m_pXSolution : double*
	m_CallCount : int
	IsLess()
	DetermineEnterVariable()
	DetermineLeaveVariable()
	UpdateInverseB()
	AdjustBasicNonbasic()
	GetMatrixAValue()
	HandleZeroCoefficient()
	<<virtual>> ~CSimplex()
	CSimplex()
	GetCurrentSolution()
	GenerateResult()
	SetParameter()
	RunLP()
	ChangeCostCoefficient()
	RecalculateInverseB()
	GetInverseB()

Figure 5.23. Class: CSimplex.

CMaxmin: getting the min/max value of the correlation formula.



















































CMaxmin	
	(m_pPenaltyFunctionGrad)(double*, int) : double
	(m_pPenaltyFunctionValue)(int*) : double
	(m_pObjectFunctionGrad)(double*) : double
	(m_pObjectFunctionValue)() : double
	m_pRhoInitialValue : double*
	m_pRhoHighBound : double*
	m_pRhoLowBound : double*
	m_pYInitialValue : double*
	m_pXInitialValue : double*
	m_pYLowBound : double*
	m_pXLowBound : double*
	m_pYHighBound : double*
	m_pXHighBound : double*
	m_RhoCondition : char
	m_DimensionofX : int
	m_DimensionofY : int
	m_pYCoefficient : double*
	m_pXCoefficient : double*
	m_pCoefficient : double*
	m_plInitialValue : double*
	m_pLowBound : double*
	m_pHighBound : double*
	m_TotalDimension : int
	CalculateXMean()
	CalculateYMean()
	CalculateXVariance()
	CalculateYVariance()
	FunctionValue()
	FunctionGrad()
	PenaltyFunctionValue()
	PenaltyFunctionGrad()
	ReverseFunctionValue()
	ReverseFunctionGrad()
	ReversePenaltyValue()
	ReversePenaltyGrad()
	VarianceValue()
	VarianceGrad()
	VariancePenlaty()
	ReverseVarianceValue()
	ReverseVarianceGrad()
	SetParameter()
	GetMin()
	GetMax()
	SetInitialValue()
	GetSolution()
	GetVarianceMin()
	GetVarianceMax()
	Optimization()
	<<virtual>> ~CMaxmin()
	CMaxmin()

Figure 5.24. Class CMaxmin.

CBoundCorrelation: getting the possible correlation range for two operands.




























CBoundCorrelation	
	m_Rows : int
	m_Columns : int
	m_pCost : double*
	(*m_pX)[3] : float
	(*m_pY)[3] : float
	m_NumberofX : int
	m_NumberofY : int
	m_pb : double*
	m_pMatrixA : double*
	m_FunctionXBound[2] : double
	m_FunctionYBound[2] : double
	m_ErrorCode : int
	m_YV[2] : double
	m_YE[2] : double
	m_XV[2] : double
	m_XE[2] : double
	m_BoundEXY[2] : double
	m_BoundCorrelation[2] : double
	MultiplyInterval()
	GetMultiplyBound()
	InitialMaxmin()
	FindBoundEXY()
	FindBoundCorrelation()
	<<virtual>> ~CBoundCorrelation()
	CBoundCorrelation()
	SetParameter()
	FindXYEVValue()

Figure 5.25. Class: CBoundCorrelation.

CMinCorrelation: getting the minimum value for specified correlation setting.

































CMinCorrelation	
	m_Rows : int
	m_Columns : int
	m_MeanX[2] : double
	m_MeanY[2] : double
	m_DXY[2] : double
	m_AdditionalConstraints : int
	m_pCost : double*
	(*m_pX)[3] : float
	(*m_pY)[3] : float
	m_NumberofX : int
	m_NumberofY : int
	m_pb : double*
	m_pMatrixA : double*
	m_LowCorrelation : double
	m_HighCorrelation : double
	m_pVariableLowBound : double*
	m_pVariableHighBound : double*
	m_FunctionXBound[2] : double
	m_FunctionYBound[2] : double
	m_ErrorCode : int
	m_IsCorrelation : int
	m_MinValue : double
	MultiplyInterval()
	CalculateMean()
	GetMultiplyBound()
	CalculateFunctionBound()
	InitialMaxmin()
	<<virtual>> ~CMinCorrelation()
	CMinCorrelation()
	SetParameter()
	RunforMin()
	ContinueRunforMin()

Figure 5.26. Class: CMinCorrelation.

CMinCorrelationExy: getting the minimum value for specified correlation or expectation of XY setting.

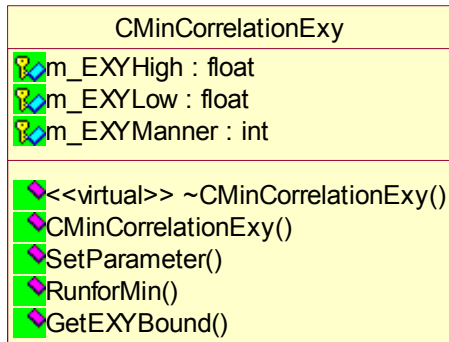


Figure 5.27. Class: CMinCorrelationExy.

### 5.5.4.2 Class relationships

In these classes, classes CBoundCorrelation, CMinCorrelation and CMinCorrelationExy are control and boundary classes, which interface with other packages or the caller and also manage the flow of logic. Classes CConfig, COptimalMin, and CSimplex are utility classes, which solve a specified problem. Classes CMaxmin and CVariance the specified functions. The following figure shows the relationship between these classes.

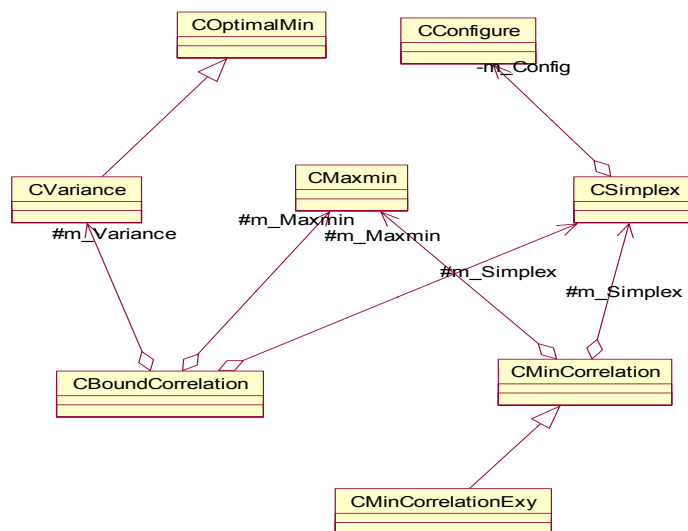


Figure 5.28. Class diagram.



### 5.5.4.3 Sequence diagrams

In this package, there are 4 interfaces: three are used to find the minimum value based on the correlation setting, and the last one is used to find the possible range of correlation.

Interface Cor\_Bound is implemented in function Cor\_Bound in file cor\_min.cpp. The following figure shows the sequence diagram:

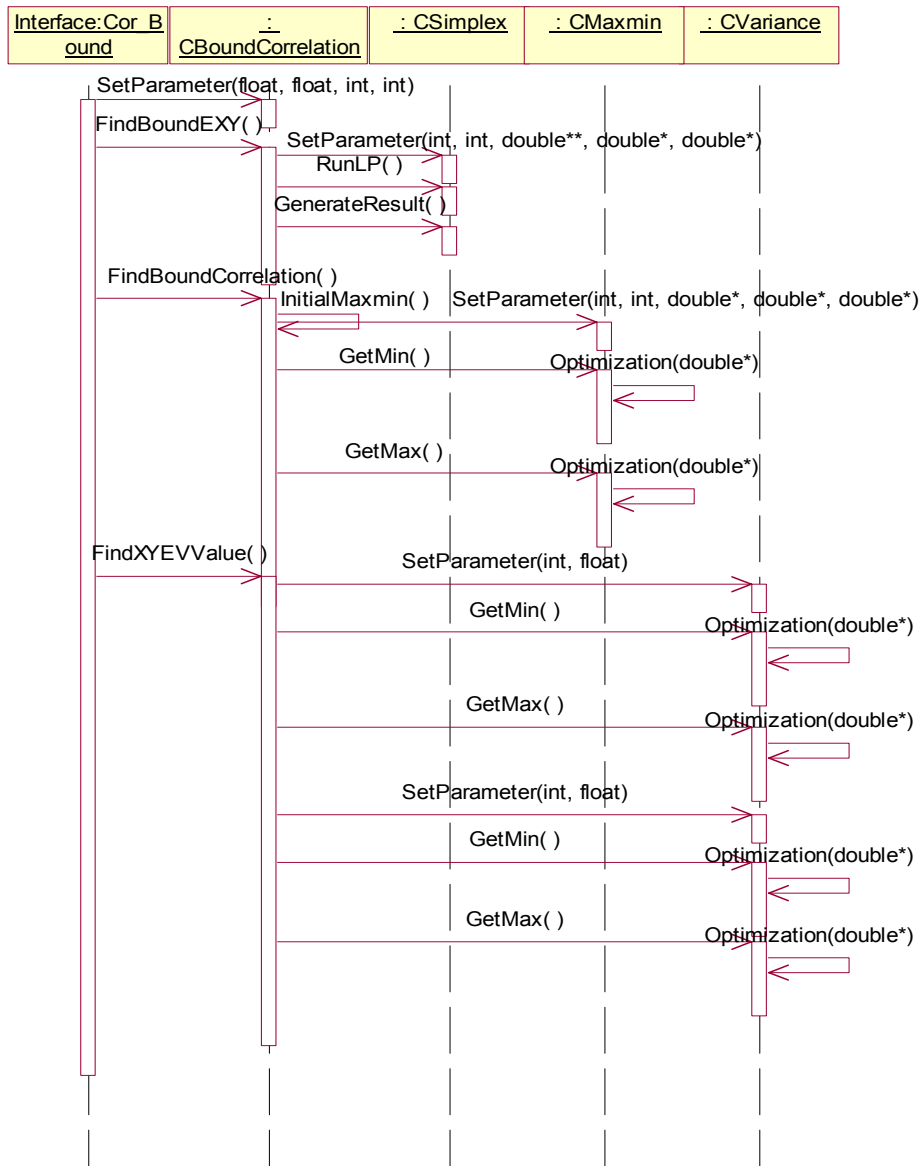


Figure 5.29. Sequence diagram for Cor\_Bound.

As previously said this software supports 3 types of correlation information setting: direct correlation, EXY setting, and X\*Y information setting. Interfaces Cor\_Min, Cor\_Min\_Exy and Cor\_Min\_Exy\_S, are implemented in file cor\_min.cpp. Cor\_Min is the simplest one of three, and supports only setting the correlation. The sequence diagram is shown as follows:



Figure 5.30. Sequence diagram for Cor\_Min.



Cor\_Min\_Exy\_S is a state-retaining version of Cor\_Min\_Exy. In this interface, a call may use information from a previous call to speed computation time. The following figure shows the sequence diagram.

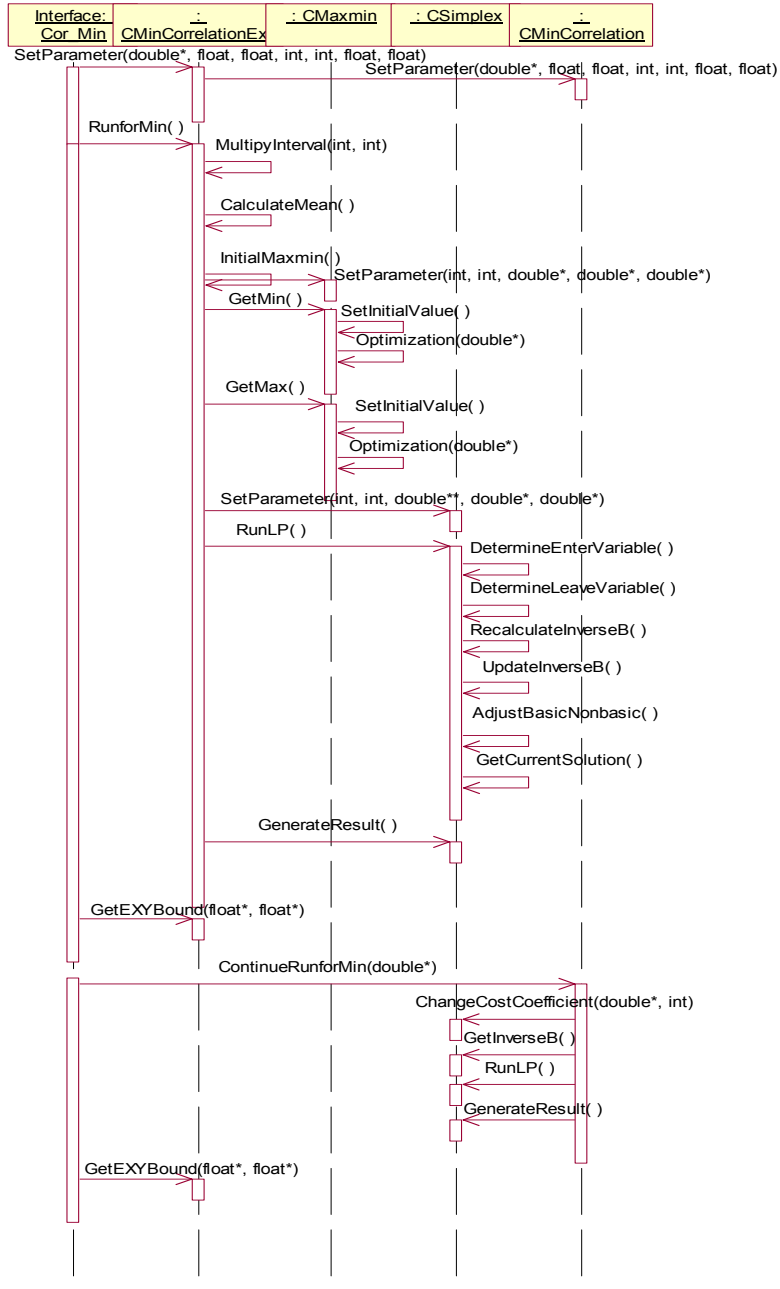


Figure 5.32. Sequence diagram for Cor\_Min\_Exy\_S.

## 6 Explanation of the results

All our experiments were conducted on the compiled version of Statool and DLLs using Visual Basic 6.0 and Visual C++ 6.0. The running platform was windows 2000 professional. The machine had 256M memory and the CPU ran at 1000Mhz.

Our experiments focus on checking three issues: the accuracy of results, the effect of correlation, and speed. Changing the accuracy of operands will affect the accuracy of results. Different correlations will change the shapes of results. Increasing the number of intervals will take more time to compute.

### 6.1 Experiments

The operand X, a random variable, was given a uniform distribution from 1 to 9. The operand Y, another random variable, was given a tail-trimmed normal distribution from 2 to 10, whose mean was 6 and variance 1. This range almost covers all the probability for Y. A small amount in the tail was omitted. We discretized the supports of X and Y into 16, 32, 64 intervals, then used the discretized X and Y as the inputs to operations. Results of operations showed the accuracy changing for different discretizations. At the same time, correlation was set to different values to check the effects. 4 operations were executed in these experiments. They are plus, minus, multiply, and divide.

The following figures show the results for different number of intervals in the operand discretizations when doing addition of X and Y with correlation zero.

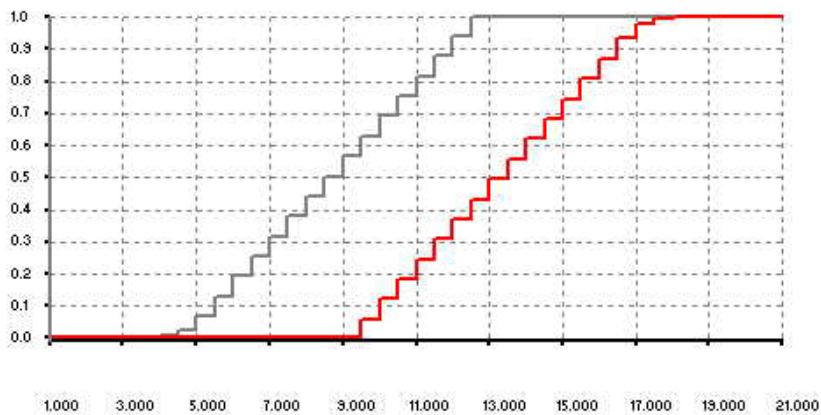
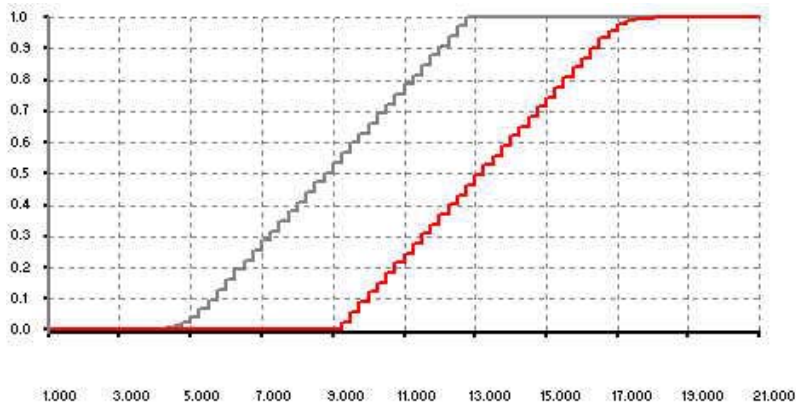
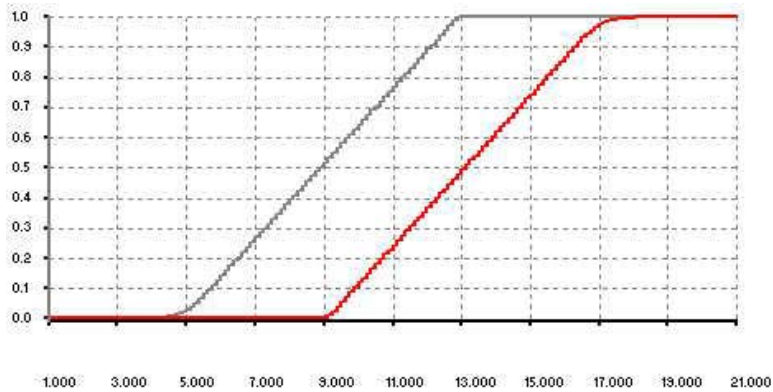


Figure 6.1.  $X+Y$  when X and Y are 16 intervals.



**Figure 6.2.**  $X+Y$  when  $X$  and  $Y$  are 32 intervals.



**Figure 6.3.**  $X+Y$  when  $X$  and  $Y$  have 64 intervals.

From these three figures, it is clear the results will become better when the discretization of the operands is increased.

Next, we show figures illustrating the effect of correlation. For this case, we let  $X$  and  $Y$  have 64 intervals, and set correlation to four values: unknown, 0.98, 0, and -0.98.

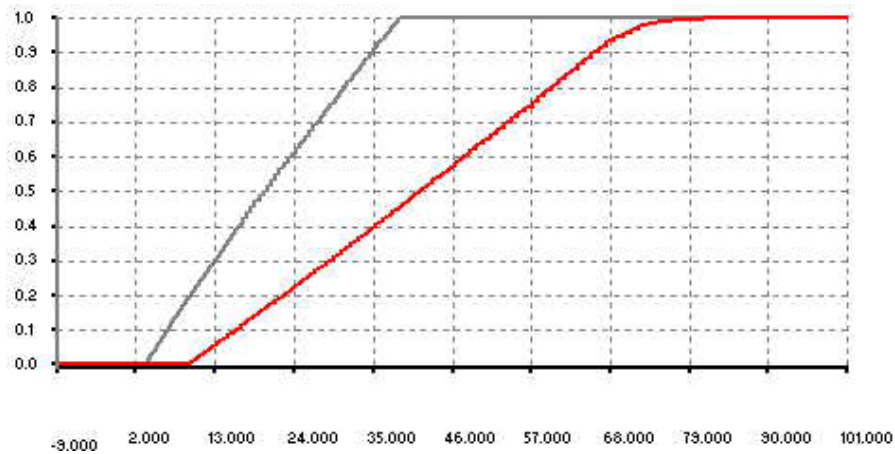


Figure 6.4.  $X*Y$  for unknown.

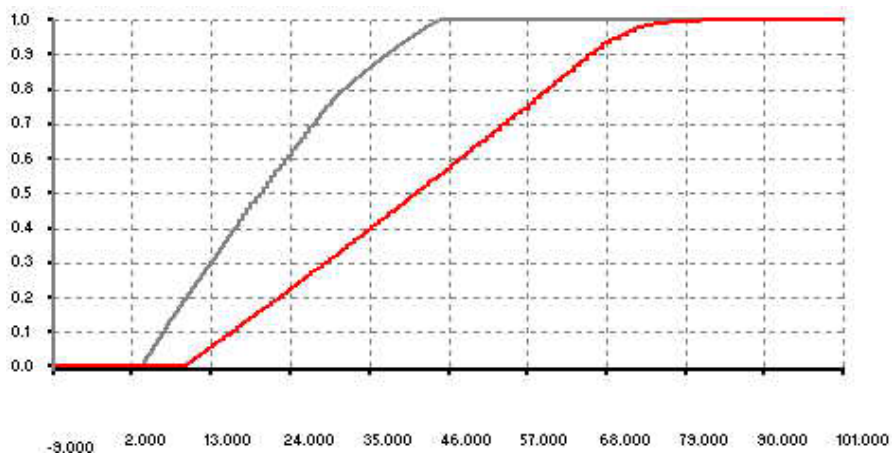


Figure 6.5.  $X*Y$  for correlation 0.98.

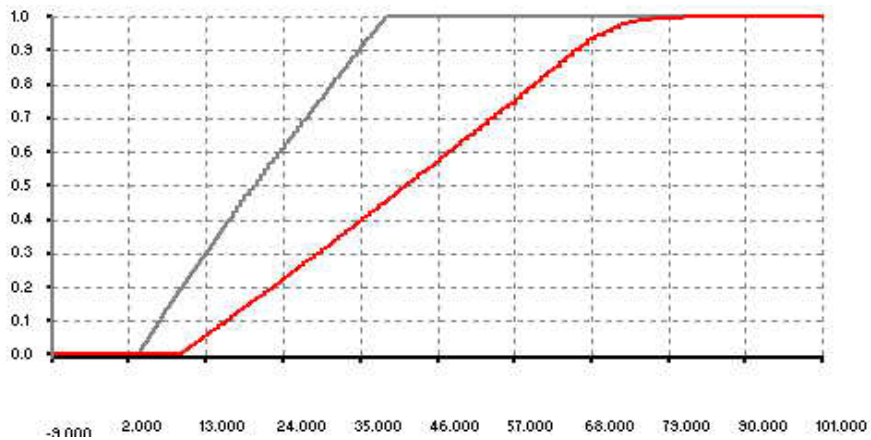
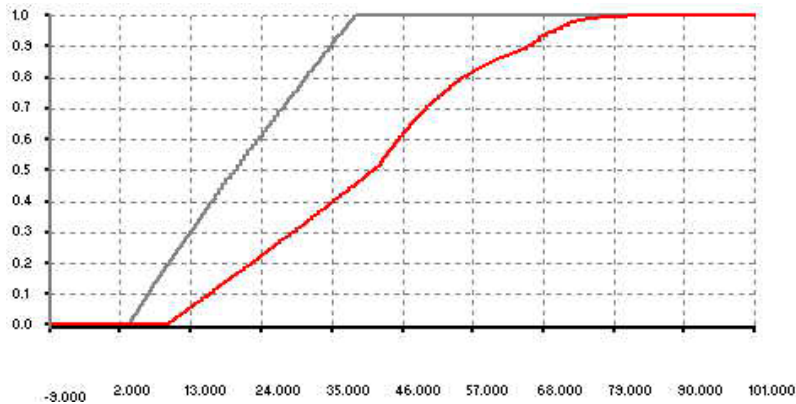


Figure 6.6.  $X*Y$  for correlation 0.



**Figure 6.7.  $X*Y$  for correlation -0.98.**

From these figures, bounds of curves can be affected by the correlation. For unknown correlation, the widest bound curves will be gotten. Compared with correlation 0, the high bound curve for correlation 0.98 is changed and the low bound curve for correlation -0.98 is changed.

The computing speed is also a factor to be considered. We did the different operations for different discretizations. We checked whether the different operations would affect the speed and what was the relationship among the computing times for the different discretizations.

**Table 6.1. Operation evaluation time (seconds) for correlation 0.**

Intervals in discretization (X x Y)	addition	subtraction	multiplication	division	max	min
16x16	1	1	3	5	1	1
32x32	22	26	154	328	13	11
64x64	3636	3297	52317	148173	1083	866

From this table, the times for plus, subtraction, max and min have the same level. Operations for multiplication and division will cost more time. Especially, division will cost 2 times multiplication. The following figure shows the times for operations: addition, subtraction, max and min.



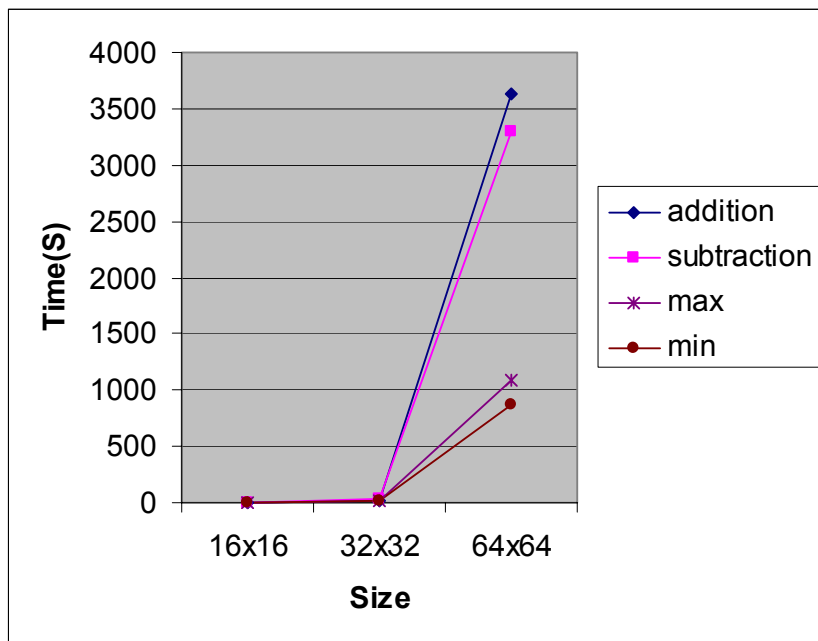


Figure 6.8. Times for operations.

This figure shows the times for multiplication and division.

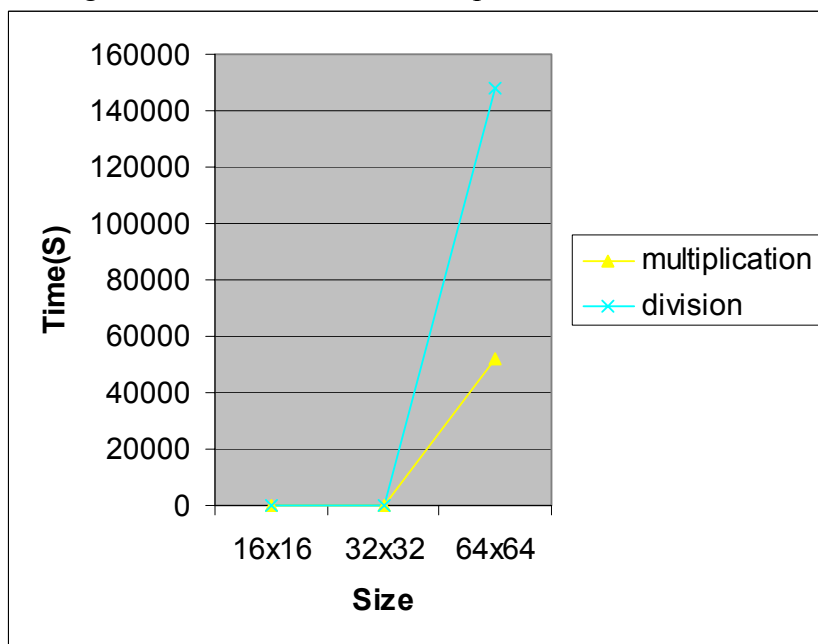


Figure 6.9. Times for multiplication and division.

## **6.2 Discussion**

Two improvements will be considered in the future: getting narrower bounds from correlation and decreasing the computing time for discretizations with many intervals. Linear programming problems have more than 4000 variables for a 64x64 discretization. So it is a big problem for linear programming. It is possible to decrease computing time if another linear programming method, whose speed is faster, or a parallel algorithm.

## References

1. D., Berleant, Automatically verified reasoning with both intervals and probability density functions, *Interval Computations* (1993 No. 2) 48-70
2. D., Berleant, Automatically verified arithmetic on probability distributions and intervals, in B. Kearfott and V. Kreinovich, eds., *Applications of Interval Computations*, Kluwer Academic Publishers, 1996, pp. 227-244
3. D., Berleant, and C. Goodman-Strauss, Bounding the results of arithmetic operations on random variables of unknown dependency using intervals, *Reliable Computing* 4 (2) (1998), pp. 147-165
4. D., Berleant, The Thickets Approach to P-Bounds, Manuscript
5. Williamson, R. & Downs, T. Probabilities arithmetic I: numerical methods for calculating convolutions and dependency bounds, *International Journal of Approximate Reasoning* 4 (1990)
6. A. Neumaier, *Interval methods for system of equations*, Cambridge University Press, 1990
7. R.E. Moore, *Methods and applications of interval analysis*, SIAM, 1979
8. N.S. Asaithambi, S. Zuhe, and R.E. Moore, On computing the range of values, *Computing*, vol.28, 1982
9. G. Alefeld, Enclosure methods in C. Ullrich, ed., *Computer arithmetic and self-validating numerical methods*, Academic Press, 1990
10. E. Hyvonen, Constraint reasoning based on interval arithmetic: the tolerance propagation approach, *Artificial Intelligence*, Vol.58, 1992
11. Ramon E. Moore, *Interval analysis*, Prentice-Hall, Inc. 1966
12. Springer, M.D., *The algebra of Random variables*, Wiley, 1979.
13. S. Ferson, What monte carlo methods cannot do, *Human and Ecological Risk Assessment* 2 (1996), pp. 990-1007
14. Frederick S. Hillier, Gerald J. Lieberman. *Introduction to operations research*, McGraw-Hill, c2001

15. Katta G. Murty. Linear and combinatorial programming, Malabar, Fla. : R.E. Krieger, 1985, c1976
16. Qian Y.Y. Operation research, Tsinghua University Press, Beijing.
17. Katta G. Murty. Robert E. Krieger, Linear and combinatorial programming, Publishing Company, 1985.
18. Vincent A. Sposito, Linear programming with statistical applications, Iowa State University Press, 1989.
19. K.-T. Fang & Y. Wang, Number-theoretic methods in statistics, 1994, Chapman & Hall
20. L.Z. Xie, Master's Thesis, University of Arkansas, 1998.
21. G.R. Walsh, Methods of optimization, 1975, John Wiley & Sons
22. G. Alefeld & J. Herzberger, Introduction to interval computations, 1983, Academic Press
23. C.S. Beightler & D.T. Phillips & D.J. Wilde, Foundations of optimization, 1979, Prentice-Hall Inc.
24. T.E. Shoup & F. Mistree, Optimizaiton methods with applications for personal computers, 1987, Prentice-Hall Inc.
25. M.J. Box & D. Davies & W.H. Swann, Non-linear optimization techniques, 1969, OLIVER & BOYD.
26. Jorage J. More & Gerardo Toraldo, On the solution of large quadratic programming problems with bound constraints, SIAM J. Optimization, Feb, 1991
27. Richard H. byrd & Peihuang Lu, A limited memory algorithm for bound constrained optimization, Technical report NAM-08, Northwestern University.
28. MatLab handbook, Online help version 6.0, 1984-2000.
29. Microsoft developer network, MSDN Library Visual Studio 6.0
30. Rational Rose, Rational unified process 5.1 online help.
31. Quatrani, Terry, Visual modeling with Rational Rose 2000 and UML, 2000, Addison Wesley.
32. Mike, Mckelvy & Ronald Martinsen & Jeff Webb, Using visual basic 5, 1997, Que.
33. Stephen R.Schach, Software engineering with Java, 1997, McGraw-Hill.