ORDER RESTRICTED A-BASIS AND B-BASIS VALUES

A Thesis by

Leo Huelskamp

Bachelor of Science, Kansas State University, 2010

Submitted to the Department of Mathematics and Statistics
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2012

© Copyright 2012 by Leo Huelskamp
All Rights Reserved

ORDER RESTRICTED A-BASIS AND B-BASIS VALUES

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Mathematics.

_____

Xiaomi Hu, Committee Chair

_____

Daowei Ma, Committee Member

_____

Chu-Ping Vijverberg, Committee Member

DEDICATION

To my dad, Terry, my mom, Mary, and my sisters, Emily, Jill, Sarah, and Meg

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Xiaomi Hu for finding me such an interesting topic to write my thesis about. He helped me immensely all along the way, from programming in C++ to helping in the background and research of the topic. He was always patient with me and motivated me throughout the process. I don't know as though I could have completed this thesis without his guidance and aide.

I would also like to thank Dr. Daowei Ma and Dr. Chu-Ping Vijverberg for giving up some of their time to be part of my thesis committee. I know how valuable their time is, and am thankful they were willing to be part of my committee.

Finally, I would like to thank my family and all of my friends for being supportive and encouraging throughout the entire process.

ABSTRACT

In this thesis, the concept of A-basis and B-basis as it relates to the aviation industry is studied. The first assumption that will be made in order to help with the formulation of the A-basis and B-basis values is that each of the samples from the obtained data is given by a normal distribution. With this assumption, the paper begins by deriving the formulas needed for obtaining A-basis and B-basis values. Once these formulas have been derived, they will be put to use in programs that are capable of running simulations. Then the thesis will examine the results and investigate if any problem areas arise. It will be discovered that there is a problem with the formulas derived as it pertains to the aviation industry, so the next step will be to find where the error originates from and find a suitable way to fix this error. The next item that will be looked at is how to successfully run simulations where it is desirable to impose certain restrictions on the population mean of each factor that is being taken into consideration. The thesis will conclude by looking at what happens when combining both the method used to fix the error discovered earlier and the method used to apply the order restriction, and again will run simulations by means of programs that take both methods into account. The results show how order restriction can be applied to the population means of the breaking strength of composite materials to obtain results that follow certain theories that have been determined by engineers.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

1.1 Motivation

In the aviation industry it is crucial that every part of a plane is able to maintain what is expected and required of it in order to keep the aircraft safe to fly. This goes from making sure the electrical components don't malfunction to ensuring the landing gears come down every time so that the aircraft is capable of landing safely. The part of the infrastructure that will be examined in this paper are the composite materials that are used in the manufacturing of the aircraft. It is vital that these composite materials are able to withstand a broad spectrum of conditions; for instance, they need to be able to withstand various temperatures and moisture levels, as well as various combinations of the two.

Through A-basis and B-basis calculations, it is possible to bring all these various testing conditions together and still be able to determine the strength that each one of the composite materials possesses. The Composite Materials Handbook, within the Military Handbook 17 F, defines A-basis and B-basis as the 95 percent lower confidence bound on the first and tenth percentile of a specified population of measurements, with specific interest in aviation of the measurements of compressive strength, shear strength, and tensile strength of composite materials [1]. In [2] the paper finds that it is possible to look at these different types of strengths under different fixed factors like temperature, layup, and humidity.

1.2 Literature Survey

Since safety is always an issue when considering the aviation industry, the Department of Defense gave a few concepts and guidelines regarding composite materials in the Military Handbook 17 F [1]. These guidelines provided a need, a need for being able to efficiently and effectively find values of the A-basis and B-basis using statistics. This led to the investigation of different ways to find the tolerance limits that were desired. So first, research was done in the area of one-sided tolerance limits and how to effectively calculate them. Much

research has been done in this field, like that of K. Krishnamoorthy and Thomas Mathew, who investigated replacing the unknown variance ratio with a generalized confidence interval [3].

Research into what type of distribution should be used in modeling the sample sets has also been done. In [4], the paper examines the 3-parameter Weibull distribution along with a Pearson, Type III distribution to analyze what happens to the tolerance limit. It was discovered later in [4] that some of the data sets that were collected did not conform to either of these two distributions. Thus they determined that a skewed distribution will not always be useful when finding A-basis and B-basis values [4]. This is one of the reasons normal distributions are still being used when evaluating the tolerance limits for A-basis and B-basis.

In order to make calculations faster, software has been developed that helps in determining A-basis and B-basis values. An early example is provided by Vangel in his paper [2], where he uses the program RECIPE to help solve one sided tolerance limits. Another software is that of STAT-17 [5], which was developed to follow the guidelines and ideas that were set forth by the Military Handbook 17 F [1]. This particular software computes the A-basis and B-basis using only a single sample, and thus is a little limited as to what it can be used for in practice. The National Center for Advanced Materials Performance has also created their own software that uses the ANOVA approach, which is called ASAP [6]. There is currently a wide-spread amount of research being done in finding A-basis and B-basis values, from the type of distribution to use to the continued development of more efficient and effective software.

1.3 Overview of Thesis

This thesis will start by deriving the formulas for each of the A-basis and B-basis values using an ANOVA approach. After that, it will look at the results of a couple simulations to see what is happening to the A-basis and B-basis values. Then the paper will look at the

inconsistency of $\overline{x}$ and basis values by running additional simulations with specific conditions predefined. Looking at how to correct these inconsistencies will be achieved by examining the original formulas and deciding what is causing these inconsistencies to transpire. Finally, after placing an order restriction on μ through the use of an order restricted convex cone, the thesis will show how this order restriction can be employed in context to composite materials in the aviation industry.

CHAPTER 2

DERIVATION OF A-BASIS AND B-BASIS VALUES

2.1 Background

One common area where A-basis and B-basis values are used is in the aviation composite material industry, where the A-basis finds the lower limit of the one-sided confidence interval of the first percentile of the breaking strength of a composite material with confidence coefficient of 95 percent, while the B-basis finds the lower limit of the one-sided confidence interval of the tenth percentile of the breaking strength of a composite material with confidence coefficient of 95 percent [1]. What is desired in this paper is to be able to take multiple factors into account while finding the A-basis and B-basis values of various composite materials. Thus, building an appropriate model is the first step that must be accomplished in order to analyze the breaking strengths of composite materials.

2.2 Model

Suppose there are r factors, and each factor has a sample size (n_i). Then there will be the samples $x_{11}, ..., x_{1n_1}; ...; x_{r1}, ..., x_{rn_r}$ with factor sample means ($\overline{x}_i$), which come from the formula $\overline{x}_i = \frac{1}{n_i}\Sigma_{j=1}^{n_i} x_{ij}$ for $i = 1$ to r. Each factor will have a sample variance (s_i^2), for $i = 1$ to r, given by $s_i^2 = \frac{\Sigma_{j=1}^{n_i}(x_{ij}-\overline{x}_i)^2}{n_i-1} = \frac{\Sigma_{j=1}^{n_i} x_{ij}^2 - \frac{(\Sigma_{j=1}^{n_i} x_{ij})^2}{n_i}}{n_i-1} = \frac{\Sigma_{j=1}^{n_i} x_{ij}^2 - n_i\overline{x}_i^2}{n_i-1}$. Next, it must be assumed that each of the factors is normally distributed with population mean μ_i and population variance σ^2, such that each factor has the same variance. With these assumption, a one-way ANOVA model can be formed. Each μ_i will be estimated by its maximum likelihood estimator, $\overline{x}_i$, which is also the unbiased estimator. Since the assumption was made that each of the samples is normal with the same variance for each factor, a common variance must be found that can be used for all samples, which is done by finding the estimator for σ^2. This estimator turns out to be an unbiased estimator which is given by $S_p^2 = \frac{\Sigma_{j=1}^{n_1}(x_{1j}-\overline{x}_1)^2+...+\Sigma_{j=1}^{n_r}(x_{rj}-\overline{x}_r)^2}{(n_1-1)+...+(n_r-1)} = \frac{(n_1-1)s_1^2+...+(n_r-1)s_r^2}{N-r}$, where N is the sum of the r sample sizes.

By Section 2 in [7], since $\overline{x}_i \sim N(\mu_i, \frac{\sigma^2}{n_i})$ and $S_p^2 \sim \chi^2(n-r)\frac{\sigma^2}{N-r}$ are independent, the formula for the A-basis and B-basis values of the ith population are defined as

$$A_i = \overline{x}_i - k_A(d_i, f)S_p \tag{1}$$

and

$$B_i = \overline{x}_i - k_B(d_i, f)S_p \tag{2}$$

where $d_i = n_i$ and $f = N - r$.

In theory it has been found that $k_A(d,f) = \frac{t_{0.95}(2.32635\sqrt{d},f)}{\sqrt{d}}$ and $k_B(d,f) = \frac{t_{0.95}(1.28155\sqrt{d},f)}{\sqrt{d}}$ [1]. For the computations that will be made in this paper, the formulas given by Tomblin, Ng and Raju in [8] will be used and are given as followed, letting $x = \frac{1}{\sqrt{f}}$ and

$$\begin{aligned}
\mathrm{q} &= 1 - 2.327x + 1.138x^2 + 0.6057x^3 - 0.3287x^4 \\
\mathrm{b}_A &= 2.0643x - 0.95145x^2 + 0.51251x^3 \\
\mathrm{b}_B &= 1.1372x - 0.49162x^2 + 0.18612x^3 \\
\mathrm{c}_A &= 0.36961 + 0.0026958x - 0.65201x^2 + 0.011320x^3 \\
\mathrm{c}_B &= 0.36961 + 0.0040342x - 0.71750x^2 + 0.16963x^3.
\end{aligned}$$

Using the equations from above, [8] finds that the values of k_A and k_B can now be approximated by

$$k_A(d,f) = \frac{2.32635}{\sqrt{q}} + \sqrt{\frac{1}{dc_A} + (\frac{b_A}{2c_A})^2} - \frac{b_A}{2c_A} \tag{3}$$

and

$$k_B(d,f) = \frac{1.28155}{\sqrt{q}} + \sqrt{\frac{1}{dc_B} + (\frac{b_B}{2c_B})^2} - \frac{b_B}{2c_B}. \tag{4}$$

Using (1) and (2) combined with these approximations, program AB from the appendix will be able to calculate A-basis and B-basis values. This program will find A-basis and B-basis values given a specific number of factors, sample size for each factor, and sample data.

Table 1: AB Results

S-Mean	89.0568	90.8213	91.1382	90.8556	93.2605	94.9812	96.7066	97.5341
S-stdev	2.9411	2.5574	4.4329	1.4106	2.9103	2.5238	2.7750	2.5895
S-Size	10	8	9	8	6	7	9	12
A:	80.3122	81.9237	82.3241	81.9580	84.1315	85.9813	87.8925	88.8989
B:	83.6230	85.2170	85.6265	85.2513	87.4034	89.2645	91.1949	92.2246

Thus, finding values for A-basis and B-basis of a sample with r factors using (1) and (2) combined with (3) and (4) from above can now be done rather quickly using software such as program AB. Unfortunately, it doesn't take long for a key issue to arise that must be taken into account and fixed in order to continue using this current approach for finding the A-basis and B-basis values for the breaking strengths of composite materials.

CHAPTER 3

INCONSISTENCY OF $\overline{x}$ AND BASIS VALUES

3.1 Issue with Sample Sizes

One of the problems that can be found when evaluating the A-basis and B-basis stems from the fact that if $\overline{x}_i < \overline{x}_j$, then the expectations are for $A_i \leq A_j$ and $B_i \leq B_j$. This comes from the fact that if the strength of a certain composite material is less than that of another composite material, it would be anticipated that the 95 percent lower confidence bound on the first and tenth percentile of the strength for the first composite material would be less than that of the second composite material. Hence, it is expected that $A_i \leq A_j$ and $B_i \leq B_j$ when $\overline{x}_i < \overline{x}_j$. While this concept does hold true in Table 1, this desired result will unfortunately not always hold. For example, when running the program AB again with a different set of input data, the following results were found.

Table 2: AB Inconsistent-Type Results

S-Mean	91.0363	89.6649	90.7342	92.0962	94.6737	94.1685	94.8892	97.4683
S-stdev	2.5988	3.3442	2.7465	1.8927	4.1156	2.7691	3.4732	3.7300
S-Size	16	8	35	8	6	25	5	4
A:	82.8042	80.9998	82.8048	83.4312	85.7690	86.1288	85.8109	88.1520
B:	86.0193	84.1748	86.0711	86.6061	88.9295	89.3727	88.9629	91.2945

In terms of the sixth and seventh factors, Table 2 shows that $\overline{x}_6 < \overline{x}_7$, which would give the implications that $A_6 \leq A_7$ and $B_6 \leq B_7$, yet the table gives $86.11288 > 85.8109$ and $89.3727 > 88.9629$ for A_6, A_7 and B_6, B_7 respectively. The reason behind this draws from the fact that there can be different factor sample sizes, as can be seen in the above simulation since there is a large difference between the sample size of the sixth factor which is 25 and the seventh which is 5. This can also be seen when examining equations (3) and (4), for

if there is a large reduction between factor sample sizes then d will decrease dramatically causing the values of (3) and (4) to actually increase. Applying this increase in (3) and (4) to equations (1) and (2) will actually result in a decrease in A_i and B_i, which makes it possible for $A_i > A_j$ and $B_i > B_j$ even though $\overline{x}_i < \overline{x}_j$. Thus, instead of using $d_i = n_i$ when calculating k_A and k_B, the averages of n_1,...,n_r will be substituted in for the value of d_i. The averages that will be examined for the replacement of each factor sample size are the p-general where p= -2, arithmetic, geometric, and harmonic averages.

3.2 Using Averages

In order to be able to use the four types of averages, it is necessary to have the formulas to find each average. Thus, these four averages are found by:

arithmetic, $\overline{n}_1 = \frac{n_1+...+n_r}{r}$,

geometric, $\overline{n}_2 = (n_1 \times \cdots \times n_r)^{1/r}$,

harmonic, $\overline{n}_3 = \frac{r}{\frac{1}{n_1}+...+\frac{1}{n_r}}$, and

p-general, $\overline{n}_4 = [(n_1^p + ... + n_r^p)/r]^{1/p}$ with p= -2.

Then to find the A-basis and B-basis values, the new formulas will be given by

$$A_i = \overline{x}_i - k_A(\overline{n}_k, f) \tag{5}$$

and

$$B_i = \overline{x}_i - k_B(\overline{n}_k, f) \tag{6}$$

for k=1,2,3,4. These equations are very similar to (1) and (2) except that now $d_i = \overline{n}_k$ instead of n_i .

Running a new program which substitutes these averages in for d_i instead of n_i, which is program AB2 in the appendix, will help in the investigation of what happens to the A-basis and B-basis values even if there is a large difference between sample sizes for each factor.

Table 3: AB2 Results

S-Mean	90.0460	91.4601	91.0790	94.1699	96.1897	100.1407
S-stdev	8.8930	3.7436	7.6191	5.9368	6.7013	6.6869
S-Size	35	5	20	20	15	20
	A-basis					
ANOVA	69.7039	68.1271	70.2142	73.3050	74.9687	79.2758
A-average	69.1326	70.5467	70.1656	73.2565	75.2763	79.2273
G-average	68.9589	70.3730	69.9919	73.0828	75.1026	79.0536
H-average	68.6747	70.0888	69.7077	72.7986	74.8184	78.7694
P-average	68.3094	69.7235	69.3424	72.4333	74.4531	78.4041
	B-basis					
ANOVA	78.0689	76.2165	78.4887	81.5795	83.1991	87.5503
A-average	77.4004	78.8145	78.4334	81.5243	83.5441	87.4951
G-average	77.2046	78.6187	78.2377	81.3285	83.3483	87.2993
H-average	76.8893	78.3034	77.9223	81.0131	83.0330	86.9840
P-average	76.4909	77.9050	77.5239	80.6148	82.6346	86.5856

Since the arithmetic average is the p-general average with p= 1, the geometric average is the limit of the p-general average when p goes to zero, and the harmonic average is the p-general average with p= -1, the following relation is found when using the p-general average where p= -2:

$$\text{arithmetic average} \geq \text{geometric average} \geq \text{harmonic average} \geq \text{p-average}.$$

These relationships between the averages also appear in the values of the A-basis and B-basis, for in Table 3 the following occurs:

$$A_{A-average} \geq A_{G-average} \geq A_{H-average} \geq A_{P-average}$$

and

$$B_{A-average} \geq B_{G-average} \geq B_{H-average} \geq B_{P-average}.$$

The other important part of this simulation can be found by looking at the second and third factors. Table 3 shows that $\overline{x}_2 > \overline{x}_3$ and yet the ANOVA model gives that $A_2 < A_3$ and $B_2 < B_3$. Once that d_i has been replaced by one of the averages, the simulation shows that this inconsistency has been corrected and always yields $A_2 \geq A_3$ and $B_2 \geq B_3$ for all four types of averages, which is exactly what was sought. Running program AB2 additional times yields the same results, that is, if $\overline{x}_i < \overline{x}_j$ then $A_i \leq A_j$ and $B_i \leq B_j$ was always true when using one of the four averages for the value of d_i.

3.3 Confidence Coefficient of Basis Values Using Averages

In the initial set-up, the goal was to find the lower limit of the one-sided interval for the first and tenth percentile with confidence coefficient of 95 percent. So the next goal is to identify which average gives the closest approximation such that it has the smallest confidence coefficient greater than or equal to 95 percent. The program CAB, from the appendix, will aide in this process by running 5000 simulations for each of the averages, and then determine what percent of these 5000 simulations lie outside the first and tenth percentile.

Table 4: CAB Results

Mu	90.50	91.00	92.20	93.00	95.00	95.60
Common sigma	3.00					
1st percentile	83.52	84.02	85.22	86.02	88.02	88.62
10-percentile	86.66	87.16	88.36	89.16	91.16	91.76
	CC for A-basis					
ANOVA	0.96	0.95	0.95	0.95	0.96	0.95
Average Size						
A-average	0.93	0.93	0.93	0.93	0.93	0.93
G-average	0.94	0.94	0.94	0.95	0.94	0.94
H-average	0.96	0.96	0.95	0.95	0.95	0.96
P-average	0.96	0.97	0.97	0.96	0.96	0.96
	CC for B-basis					
ANOVA	0.96	0.95	0.96	0.95	0.96	0.95
Average Size						
A-average	0.92	0.92	0.92	0.92	0.93	0.93
G-average	0.94	0.94	0.94	0.95	0.94	0.94
H-average	0.96	0.96	0.95	0.95	0.95	0.96
P-average	0.96	0.97	0.97	0.96	0.96	0.97

Based on the inequality of the averages, the best place to start looking is at the A-basis and B-basis associated with the arithmetic average. Since the arithmetic average has values all below 0.95 it can be concluded that the arithmetic average will not give the desired 95 percent confidence coefficient. Examining the A-basis and B-basis values resulting from the geometric average yields the same conclusion, for there are multiple values below the desired 0.95. Continuing on with the A-basis and B-basis associated with the harmonic average, Table 4 shows that each value is at least 0.95. Running program CAB numerous times gives

the same results each time with each value in the rows of H-average being at least 0.95. Thus, the harmonic average is the average to use in order to get a confidence coefficient of at least 95 percent. Thus, on future simulations the harmonic average will be used for the value of d_i, which will correct the inconsistency that was found between $\overline{x}$ and the basis values in Table 2.

CHAPTER 4

IMPOSING ORDER RESTRICTION ON μ

4.1 Fixed Factors

In [2], the author mentions the topic of testing being expensive the more data that is collected and how it is ideal to run a small number of samples if possible. Thus, [2] describes the ideal situation would be to have data on strengths available under different fixed factors and find the A-basis and B-basis values on all the results together. By grouping all the data into one set, it is possible to look at the means of each strength under each fixed factor. For instance, it may be desirable to find the A-basis and B-basis values of tensile strength under different temperatures or humidities, which are both fixed factors. However, there may be additional information that engineers may already know about the tensile, compressive, or shear strengths of a composite under these particular fixed factors that will need to be taken into account.

In [9], the authors were curious whether the mechanical properties of composite materials would be affected if some of these fixed factors were changed. By the end of the paper, they discovered that for 90° laminates of Thornel 300/Fiberite 1034 graphite epoxy composites an increase in temperature from 200K to 450K created a significant decrease in tensile strength [9]. Thus, it would be expected that the population mean for the tensile strength of the 90° laminate under temperature of 200K would be greater than that of the 90° laminate under temperature of 450K.

4.2 Order Restriction

In order to assure potential inequalities hold true for the population means, an order restriction must be placed on their maximum likelihood estimators, which are the $\overline{x}_i$'s, for i=1,2,...,r. For instance, in order to satisfy the order restriction $\mu_1 \leq \mu_2 \leq ... \leq \mu_r$, it is necessary to impose the order restriction onto their maximum likelihood estimators such that $\tilde{x}_1 \leq \tilde{x}_2 \leq ... \leq \tilde{x}_r$, where $\tilde{x}_i$ is the restricted maximum likelihood estimator. In order

to make this specific order restriction work, projection of the vector $M = (\mu_1, \mu_2, ..., \mu_r)$ onto the order restricted convex cone satisfying the desired order with respect to the metric, which is defined by a weight vector $W = (n_1, n_2, ..., n_r)$ where n_i represents the sample size of the ith factor, must occur. This creates the projected vector that has the order restriction desired. For example, running the program OP obtains the following results, where f represents the μ's, w the weights, and p the projected values onto the order restricted convex cone.

Table 5: OP Results

f	90.0000	81.0000	93.0000	90.0000	97.0000	99.0000
w	3.0000	1.2000	2.0000	1.0000	5.0000	2.0000
Orders	$1 \leq 2$	$2 \leq 3$	$3 \leq 4$	$4 \leq 5$	$5 \leq 6$	
p	87.4286	87.4286	92.0000	92.0000	97.0000	99.0000

As seen in Table 5, the projected values now satisfy $p_1 \leq p_2 \leq ... \leq p_6$, which is what was expected on this specific order restricted convex cone. Using the information found in Chapter 3 along with this idea of projecting the population means onto an order restricted convex cone will make sure that both the order restriction on the population means is satisfied and that if $\mu_i < \mu_j$ then $A_i \leq A_j$ and $B_i \leq B_j$. Thus, the value of d_i will take on the value of the harmonic mean, instead of the sample size of each factor. This idea will now be integrated into the simulation in order to find the A-basis and B-basis values of a sample that may not initially satisfy the desired restricted ordering, which is done by running program AB3. Note, in Table 6 RMLE stands for restricted maximum likelihood estimator, whose values are estimators for each $\tilde{x}_i$ as well as the values of the A-basis and B-basis under the order restriction.

Table 6: AB3 Results

S-Size	35	5	20	20	15	20
S-Stdev	7.1478	8.9775	6.6861	7.5770	7.0278	6.8479
Xbar	90.4524	90.6472	94.2519	92.3525	97.3119	100.1388
RMLE	90.4524	90.6472	93.3022	93.3022	97.3119	100.1388
A-basis						
ANOVA	70.8248	68.1337	74.1198	72.2204	76.8362	80.0067
H-average	69.8317	70.0265	73.6311	71.7317	76.6911	79.5180
RMLE H-ave	69.8317	70.0265	72.6814	72.6814	76.6911	79.5180
B-basis						
ANOVA	78.8959	75.9389	82.1037	80.2042	84.7775	87.9906
H-average	77.7578	77.9526	81.5572	79.6578	84.6172	87.4441
RMLE H-ave	77.7578	77.9526	80.6075	80.6075	84.6172	87.4441
Ordering Imposed:	1 <= 2	2 <= 3	3 <= 4	4 <= 5	5 <= 6	

The order restriction that was desired in this simulation was for $\mu_1 \leq \mu_2 \leq ... \leq \mu_6$, which was achieved by projecting each sample's maximum likelihood estimator, or $\overline{x}_i$, onto the order restricted convex cone. Note that in Table 6, the corresponding A-basis and B-basis values are such that $A_i \leq A_j$ and $B_i \leq B_j$ for $i \leq j$, where the value of d_i took on the harmonic average value. This conclusion held for each simulation that was run using program AB3. These A-basis and B-basis values are the restricted maximum likelihood estimators for A_i and B_i on the order restricted convex cone. Thus, by imposing an order restriction on μ, the same order restriction will also be placed on the corresponding A-basis and B-basis values, when also using the harmonic average for the value of d_i.

4.3 Confidence Coefficients of Basis Values with Order Restriction Imposed

The final step is to look at the confidence coefficient of A-basis and B-basis with the desired order restriction by projecting each one of the sample means onto the order restricted convex cone and using the harmonic average as the value of d_i. Results on the confidence coefficient will be yielded from the simulations ran using the program FINAL, which will run 5000 simulations for each of the ANOVA case, harmonic average case, and finally the harmonic average case in which the sample means are projected onto the order restricted convex cone.

Table 7: FINAL Results

	ANOVA settings					
# of populations	6					
Common Sigma	7.00					
Means	80.00	81.00	88.00	95.00	102.00	103.00
# of Rounds	5000					
Random S-Size	$5 - 49$					
Mu	80.00	81.00	88.00	95.00	102.00	103.00
1st percentile	63.72	64.72	71.72	78.72	85.72	86.72
10-percentile	71.03	72.03	79.03	86.03	93.03	94.03
	CC for A-basis					
ANOVA	0.9548	0.9540	0.9498	0.9562	0.9518	0.9518
H-average	0.9484	0.9552	0.9534	0.9520	0.9514	0.9526
Restricted	0.9816	0.9508	0.9532	0.9522	0.9832	0.9562
	CC for B-basis					
ANOVA	0.9542	0.9514	0.9504	0.9554	0.9526	0.9504
H-average	0.9476	0.9568	0.9530	0.9562	0.9516	0.9516
Restricted	0.9898	0.9520	0.9532	0.9550	0.9872	0.9594
Restrictions						
	1 <= 2	5 <= 6				

So in this simulation, the order restricted convex cone being used is such that $\mu_1 \leq \mu_2$ and $\mu_5 \leq \mu_6$, which means that it is necessary for the maximum likelihood estimator for the population mean of the first factor to be less than that of the second, and the fifth factor to be less than that of the sixth. Thus based on the simulations run in section 4.2, it is expected

that the A-basis and B-basis values of factor one are to be less than that of factor two and for that of factor five to be less than that of factor six. These expectations do hold true, since the confidence coefficient of the first factor's A-basis and B-basis are 98.16 percent and 98.98 percent while that of the second are 95.08 percent and 95.20 percent. The same holds true when looking at factors five and six from Table 7. This simulation will work for any type of order restriction that is desired, for example, $\mu_1 \leq \mu_2 \leq ... \leq \mu_r$ could be applied instead, thus creating a new order restricted convex cone to project each $\overline{x}_i$ onto. Table 7 also illustrates how the confidence coefficient of the order restricted A-basis and B-basis all meet the desired 95 percent, which is what is expected in A-basis and B-basis values.

CHAPTER 5

CONCLUSION

The values of A-basis and B-basis play an important role in aviation production and safety. Thus, it is very important to be able to find these values at an efficient rate. After analyzing the ANOVA model given by (1) and (2), which use the sample size for each factor as the value of d_i, it was discovered that it is possible to have $A_i > A_j$ and $B_i > B_j$ even though $\overline{x}_i < \overline{x}_j$, which contradicts general expectations. To take care of this inconsistency with $\overline{x}_i$ and A-basis and B-basis values, substituting in one of the four averages for d_i must take place, which is done in equations (5) and (6). After running simulations, it was discovered that the harmonic average both corrected the inconsistencies found between $\overline{x}_i$ and the basis values and also gave probabilities closest to the desired A-basis and B-basis confidence coefficients of 95 percent.

Engineers have amassed an extensive amount of information on composite materials. Thus, data of how composites respond under many types of different fixed factors gives an impression as to what should be expected in terms of the population means for tensile, compressive, and shear strengths. In order to have the ability to take into account this knowledge of composite materials, there must be some order restrictions set in place, which is achieved using projections onto an order restricted convex cone. Through the simulations that were run, it was discovered that by applying an order restriction on the maximum likelihood estimators of the population means and by using the harmonic average of all the sample sizes, an order restriction was successfully imposed on μ as well as the A-basis and B-basis values of the factors. Thus making it possible to calculate A-basis and B-basis values that adhere to previous understandings of the composite materials being analyzed.

REFERENCES

LIST OF REFERENCES

1. Military Handbook - MIL-HDBK-17-1F: Composite Materials Handbook, Volume 1 - Polymer Matrix Composites Guidelines for Characterization of Structural Materials (2002), U.S. Department of Defense.

2. Mark G. Vangel, *A User's Guide to RECIPE: A FORTRAN Program for Determining One-Sided Tolerance Limits for Mixed Models with Two Components of Variance, Version 1.0*, National Institute of Standards and Technology Statistical Engineering Division (1994).

3. K. Krishnamoorthy and Thomas Mathew, *One-Sided Tolerance Limits in Balanced and Unbalanced One-Way Random Models Based on Generalized Confidence Intervals*, TECHNOMETRICS February 2004, Vol. 46, NO. 1.

4. Richard C. Rice, Randall J. Goode, Steven R. Thompson, and John G. Bakuckas, Jr., *Development of MMPDS Handbook Aircraft Design Allowables*, DOD/FAA/NASA Conference on Aging Aircraft (2003).

5. STAT-17 software available through the Organization of the Composite Materials Handbook 17 at http://www.cmh17.org, [Accessed March 2012].

6. ASAP software available through the National Center of Advanced Materials Performance at http://www.niar.wichita.edu/coe/ncamp.asp, [Accessed March 2012].

7. Xiaomi Hu, Asymptotical distributions, parameters and coverage probabilities of tolerance limits, *Computational Statistics and Data Analysis* (2007), **51**(9) $4753 - 4760$.

8. John S. Tomblin, Yeow C. NG and K. Sresh Raju, *Material qualification and equivalency for polymer matrix composite materials systems: Updated procedure*, DOT/FAA/AR-03/19 (2003), Federal Aviation Administration William J. Hughes Technical Center.

9. Chi-Hung Shen and George S. Springer, *Effects of Moisture and Temperature on the Tensile Strength of Composite Materials*, Department of Mechanical Engineering at The University of Michigan (1976).

APPENDIX

APPENDIX

Random Sample Code

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>/*rand()*/
#include<math.h>/*pow()*/
#define PI 3.1415926
/*******************/
double obsN(double mu, double sigma){
int i, n=40;
double r=0.0;
for(i=0;i<n;i++)
r+=(rand()%100);
r-=(49.5*n);
r/=sqrt(3333.0*n/4.0);
r*=sigma;
r+=mu;
return r;}
/*******************/
void close_down(int id){
if(id==1){
printf("\n\t Fail to open file to read!\n");
printf("\n\t Input file name: s0.txt\n");
printf("\n\t It contains");
printf("\n\t  (1) NS -number of samples");
printf("\n\t  (2) Mu[1]..Mu[NS]  -populations means");
printf("\n\t  (3) Sigma -population stdev");
printf("\n\t  (4) SIze[1]..Size[NS] -Sample sizes");}
if(id==2){
printf("\n\t Fail to open file to write! \n");
printf("\n\t Output file name s1.txt");}
if(id==0){
printf("\n\t Results saved in s1.txt");}
printf("\n\n\t Press any key to quit \n");
getch();
exit(0);}
/*******************/
FILE*infp,*outfp;
int NS;
double Mu[10], sigma;
int Size[10];
void main(void){
```

```
int i, j,k;
double prt;
clrscr();
infp=fopen("s0.txt","r");
if(infp==NULL)
close_down(1);
fscanf(infp,"%d",&NS);
for(i=1;i<=NS;i++){
fscanf(infp,"%lf",&Mu[i]);}
fscanf(infp,"%lf",&sigma);
for(i=1;i<=NS;i++)
fscanf(infp,"%d",&Size[i]);
fclose(infp);
outfp=fopen("s1.txt","w");
if(outfp==NULL)
close_down(2);
fprintf(outfp,"\n%d",NS);
fprintf(outfp,"\n\n");
for(i=1;i<=NS;i++)
fprintf(outfp,"%3d", Size[i]);
randomize();
for(i=1;i<=NS;i++){
fprintf(outfp,"\n\n");
for(j=1;j<=Size[i];j++){
prt=obsN(Mu[i],sigma);
fprintf(outfp," %9.4f ",prt);
k=j%5;
if(k==0)
fprintf(outfp,"\n");}}
fprintf(outfp,"%3d",NS);
fprintf(outfp,"%9.4f",sigma);
fprintf(outfp,"%9.4f",Mu[NS]);
fclose(outfp);
close_down(0);}
```

AB Code

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
FILE *infp, *outfp;
int NS;
int Size[11];
double xbar[11];
double stdev[11];
double Kas[11], Kbs[11], A[11], B[11];
double f, s;
/*****************************/
/*     kA(d,f) and kB(d,f)   */
/*****************************/
double p4(double a0, double a1, double a2, double a3, double a4, double x){
double r;
r=a3+a4*x;
r=a2+r*x;
r=a1+r*x;
r=a0+r*x;
return r;}
double myK(double z, double q, double d, double b, double c){
double r1,r2,r3;
r1=z/sqrt(q);
r3=0.5*b/c;
r2=1.0/d;
r2=r2/c;
r2=r2+r3*r3;
r2=sqrt(r2);
r1=r1+r2-r3;
return r1;}
double kA(double d, double f){
double x,q,b,c,k_A;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,.6057,-0.3287,x);
b=p4(0,2.0643,-0.95145,0.51251,0,x);
c=p4(0.36961,0.0026958,-0.65201,0.011320,0,x);
k_A=myK(2.32635,q,d,b,c);
return k_A;}
double kB(double d, double f){
double x,q,b,c,k_B;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,.6057,-0.3287,x);
```

```
b=p4(0,1.1372,-0.49162,0.18612,0,x);
c=p4(0.36961,0.0040342,-0.71750,0.19693,0,x);
k_B=myK(1.28155,q,d,b,c);
return k_B;}
/*****************************/
void close_down(int id){
if(id==1){
printf("\n\t Fail to open file to read!\n");
printf("\n\t Input file name: s1.txt\n");
printf("\n\t It contains");
printf("\n\t  (1) NS - number of samples");
printf("\n\t  (2) Size[1]..Size[NS] - Sample sizes");
printf("\n\t  (3) sample1..sampleNS - Samples");}
if(id==2){
printf("\n\t Fail to open file to write!\n");
printf("\n\t Output file name AB_out.txt");}
if(id==0){
printf("\n\t Results saved in AB_out.txt");}
printf("\n\n\t Press any key to quit\n");
getch();
exit(0);}
/*****************************/
void main(void){
clrscr();
int i,j;
double x,sx,sxx,rr;
/*----------------
open file, read NS, Size[], Samples
compute xbar[], stdev[], close file
----------------*/
infp=fopen("s1.txt","r");
if(infp==NULL) close_down(1);
fscanf(infp,"%d",&NS);
for(i=1;i<=NS;i++){
fscanf(infp,"%d",&Size[i]);}
for(i=1;i<=NS;i++){
sx=0;sxx=0;
for(j=1;j<=Size[i];j++){
fscanf(infp,"%lf",&x);
sx+=x;
sxx+=(x*x);}
x=(double) Size[i]; xbar[i]=sx/x;
sxx=sxx-sx*sx/x;
sxx=sxx/(x-1);
stdev[i]=sqrt(sxx);}
```

```
fclose(infp);
/*-------------------
compute f,s,Kas[],Kbs[]
-------------------*/
f=0;s=0;
for(i=1;i<=NS;i++){
x=(double) Size[i];
f+=x;
s+=stdev[i]*stdev[i]*(x-1);}
f=f-NS; s=sqrt(s/f);
for(i=1;i<=NS;i++){
Kas[i]=kA((double)Size[i],f);
Kbs[i]=kB((double)Size[i],f);
Kas[i]=Kas[i]*s;
Kbs[i]=Kbs[i]*s;}
/*------------------
compute A[],B[]
------------------*/
for(i=1;i<=NS;i++){
A[i]=xbar[i]-Kas[i];
B[i]=xbar[i]-Kbs[i];}
/*------------------
output
------------------*/
outfp=fopen("AB_out.txt","w");
if(outfp==NULL) close_down(2);
fprintf(outfp,"\n\n\t\tA-basis and B-basis\n");
fprintf(outfp,"\n\tS-Mean ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",xbar[i]);}
fprintf(outfp,"\n\tS-stdev");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",stdev[i]);}
fprintf(outfp,"\n\tS-Size");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 5d    ",Size[i]);}
fprintf(outfp,"\n\n");
fprintf(outfp,"\n\tA:      ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",A[i]);}
fprintf(outfp,"\n\tB:      ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",B[i]);}
fclose(outfp);
close_down(0);}
```

AB2 Code

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
FILE *infp, *outfp;
int NS;
int Size[11];
double xbar[11];
double stdev[11];
double Kas[11], Kbs[11], A[11], B[11], Kaas[11],Kabs[11],
Kgas[11],Kgbs[11],Khas[11],Khbs[11],KPas[11],KPbs[11];
double C[11], D[11], E[11], F[11], G[11], H[11], I[11], J[11];
double f, s,a,g,h,P;
/*****************************/
/*     kA(d,f) and kB(d,f)   */
/*****************************/
double p4(double a0, double a1, double a2, double a3, double a4, double x){
double r;
r=a3+a4*x;
r=a2+r*x;
r=a1+r*x;
r=a0+r*x;
return r;}
double myK(double z, double q, double d, double b, double c){
double r1,r2,r3;
r1=z/sqrt(q);
r3=0.5*b/c;
r2=1.0/d;
r2=r2/c;
r2=r2+r3*r3;
r2=sqrt(r2);
r1=r1+r2-r3;
return r1;}
double kA(double d, double f){
double x,q,b,c,k_A;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,.6057,-0.3287,x);
b=p4(0,2.0643,-0.95145,0.51251,0,x);
c=p4(0.36961,0.0026958,-0.65201,0.011320,0,x);
k_A=myK(2.32635,q,d,b,c);
return k_A;}
double kB(double d, double f){
double x,q,b,c,k_B;
```

```
x=sqrt(1/f);
q=p4(1,-2.327,1.138,.6057,-0.3287,x);
b=p4(0,1.1372,-0.49162,0.18612,0,x);
c=p4(0.36961,0.0040342,-0.71750,0.19693,0,x);
k_B=myK(1.28155,q,d,b,c);
return k_B;}
/*****************************/
void close_down(int id){
if(id==1){
printf("\n\t Fail to open file to read!\n");
printf("\n\t Input file name: s1.txt\n");
printf("\n\t It contains");
printf("\n\t  (1) NS - number of samples");
printf("\n\t  (2) Size[1]..Size[NS] - Sample sizes");
printf("\n\t  (3) sample1..sampleNS - Samples");}
if(id==2){
printf("\n\t Fail to open file to write!\n");
printf("\n\t Output file name AB2_out.txt");}
if(id==0){
printf("\n\t Results saved in AB2_out.txt");}
printf("\n\n\t Press any key to quit\n");
getch();
exit(0);}
/*****************************/
void main(void){
clrscr();
int i,j;
double x,sx,sxx,rr;
/*----------------
open file, read NS, Size[], Samples
compute xbar[], stdev[], close file
----------------*/
infp=fopen("s1.txt","r");
if(infp==NULL) close_down(1);
fscanf(infp,"%d",&NS);
for(i=1;i<=NS;i++){
fscanf(infp,"%d",&Size[i]);}
for(i=1;i<=NS;i++){
sx=0;
sxx=0;
for(j=1;j<=Size[i];j++){
fscanf(infp,"%lf",&x);
sx+=x;
sxx+=(x*x);}
x=(double) Size[i]; xbar[i]=sx/x;
```

```
sxx=sxx-sx*sx/x;
sxx=sxx/(x-1);
stdev[i]=sqrt(sxx);}
fclose(infp);
/*-------------------
compute averages
-------------------*/
a=0;g=1;h=0;P=0;
for(i=1;i<=NS;i++){
x=(double)Size[i];
a+=x;}
a=a/NS;
for(i=1;i<=NS;i++){
x=(double)Size[i];
g=g*x;}
double z;
z=(double)1/NS;
g=pow(g,z);
for(i=1;i<=NS;i++){
x=(double)Size[i];
h+=(1/x);}
h=NS/h;
for(i=1;i<=NS;i++){
x=(double)Size[i];
z=(double) -2;
P=pow(x,z)+P;}
P=P/NS;
z=(double) -1/2;
P=pow(P,z);
/*-------------------
compute f,s,Kas[],Kbs[]
-------------------*/
f=0;s=0;
for(i=1;i<=NS;i++){
x=(double) Size[i];
f+=x;
s+=stdev[i]*stdev[i]*(x-1);}
f=f-NS; s=sqrt(s/f);
for(i=1;i<=NS;i++){
Kas[i]=kA((double)Size[i],f);
Kbs[i]=kB((double)Size[i],f);
Kas[i]=Kas[i]*s;
Kbs[i]=Kbs[i]*s;}
double sa,sg,sh,sP;
sa=0;sg=0;sh=0,sP=0;
```

```
for(i=1;i<=NS;i++){
sa+=stdev[i]*stdev[i]*(a-1);}
sa=sqrt(sa/f);
for(i=1;i<=NS;i++){
Kaas[i]=kA(a,f);
Kabs[i]=kB(a,f);
Kaas[i]=Kaas[i]*sa;
Kabs[i]=Kabs[i]*sa;}
for(i=1;i<=NS;i++){
sg+=stdev[i]*stdev[i]*(g-1);}
sg=sqrt(sg/f);
for(i=1;i<=NS;i++){
Kgas[i]=kA(g,f);
Kgbs[i]=kB(g,f);
Kgas[i]=Kgas[i]*sg;
Kgbs[i]=Kgbs[i]*sg;}
for(i=1;i<=NS;i++){
sh+=stdev[i]*stdev[i]*(h-1);}
sh=sqrt(sh/f);
for(i=1;i<=NS;i++){
Khas[i]=kA(h,f);
Khbs[i]=kB(h,f);
Khas[i]=Khas[i]*sh;
Khbs[i]=Khbs[i]*sh;}
for(i=1;i<=NS;i++){
sP+=stdev[i]*stdev[i]*(P-1);}
sP=sqrt(sP/f);
for(i=1;i<=NS;i++){
KPas[i]=kA(P,f);
KPbs[i]=kB(P,f);
KPas[i]=KPas[i]*sP;
KPbs[i]=KPbs[i]*sP;}
/*------------------
compute A[],B[]
------------------*/
for(i=1;i<=NS;i++){
A[i]=xbar[i]-Kas[i];
B[i]=xbar[i]-Kbs[i];
C[i]=xbar[i]-Kaas[i];
D[i]=xbar[i]-Kabs[i];
E[i]=xbar[i]-Kgas[i];
F[i]=xbar[i]-Kgbs[i];
G[i]=xbar[i]-Khas[i];
H[i]=xbar[i]-Khbs[i];
I[i]=xbar[i]-KPas[i];
```

```
J[i]=xbar[i]-KPbs[i];}
/*------------------
output
------------------*/
outfp=fopen("AB2_out.txt","w");
if(outfp==NULL) close_down(2);
fprintf(outfp,"\n\n\t\tA-basis and B-basis\n");
fprintf(outfp,"\n\tS-Mean       ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",xbar[i]);}
fprintf(outfp,"\n\tS-stdev      ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",stdev[i]);}
fprintf(outfp,"\n\tS-Size       ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 5d    ",Size[i]);}
fprintf(outfp,"\n\n");
fprintf(outfp,"\n\tA-basis      ");
fprintf(outfp,"\n\tANOVA        ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",A[i]);}
fprintf(outfp,"\n\tA-average    ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",C[i]);}
fprintf(outfp,"\n\tG-average    ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",E[i]);}
fprintf(outfp,"\n\tH-average    ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",G[i]);}
fprintf(outfp,"\n\tP-average    ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",I[i]);}
fprintf(outfp,"\n\n");
fprintf(outfp,"\n\tB-basis      ");
fprintf(outfp,"\n\tANOVA        ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",B[i]);}
fprintf(outfp,"\n\tA-average    ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",D[i]);}
fprintf(outfp,"\n\tG-average    ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",F[i]);}
fprintf(outfp,"\n\tH-average    ");
```

```
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",H[i]);}
fprintf(outfp,"\n\tP-average    ");
for(i=1;i<=NS;i++){
fprintf(outfp,"% 9.4f",J[i]);}
fprintf(outfp,"\n\n");
fclose(outfp);
close_down(0);}
```

CAB Code

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAXN 25
/*----- input -----*/
FILE *infp; FILE *outfp;
int NS;
double Mu[MAXN], sigma;
/*---- computation ---*/
double P10[MAXN], P01[MAXN];
double CA[5][MAXN], CB[5][MAXN];
double Round;
int Size[MAXN];
double Xbar[MAXN], Stdev[MAXN], A, B;
double f, s;
double d[MAXN];
/*****************************************/
/********** 1. simul_stat() **********/
/*****************************************/
double obsN(double mu, double sigma){
int i, n=40; double r=0.0;
for(i=0;i<n;i++) r+=(rand()%100);
r-=(49.5*n); r/=sqrt(3333.0*n/4);
r*=sigma; r+=mu; return r; }
void simul_stat(int n, double mu, double sigma, double *Sm, double *Ss){
int i; double x, sx, sxx; sx=0; sxx=0;
for(i=1;i<=n;i++){x=obsN(mu,sigma); sx+=x; sxx+=(x*x);}
x=sx/(double)n; *Sm=x;
sx=sx*sx/(double)n; sxx=sxx-sx; sxx=sxx/(double)(n-1);
sxx=sqrt(sxx); *Ss=sxx;}
/*****************************************/
/******* 2. kA(d,f), kB(d,f) *********/
/*****************************************/
double p4(double a0, double a1, double a2, double a3, double a4, double x){
double r;
r=a3+a4*x; r=a2+r*x; r=a1+r*x; r=a0+r*x;
return r;}
double k(double z, double q, double d, double b, double c){
double r1, r2, r3;
r1=z/sqrt(q); r3=0.5*b/c; r2=1.0/d;
r2=r2/c; r2=r2+r3*r3; r2=sqrt(r2); r1=r1+r2-r3;
return r1;}
```

```
double kA(double d, double f){
double x,q,b,c, k_A;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,0.6057,-0.3287,x);
b=p4(0,2.0643,-0.95145,0.51251,0,x);
c=p4(0.36961,0.0026958,-0.65201,0.011320,0,x);
k_A=k(2.32635,q,d,b,c);
return k_A;}
double kB(double d, double f){
double x, q,b,c, k_B;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,0.6057,-0.3287,x);
b=p4(0,1.1372,-0.49162,0.18612,0,x);
c=p4(0.36961,0.0040342,-0.71750,0.19693,0,x);
k_B=k(1.28155,q,d,b,c);
return k_B;}
/***********************************************/
/* ======== 3. close_down() ==================*/
/***********************************************/
void close_down(int id){
printf("\n\n");
if(id==1){
printf("\n\t Fail to open data file\n");
printf("\n\t Data file name: CAB_in.txt\n");
printf("\n\t It contains");
printf("\n\t (1) NS --- number of samples");
printf("\n\t (2) Mu[1]..Mu[NS] - population means");
printf("\n\t (3) sigma --- population stdev");}
if(id==2){ printf("\n\t Case classification error");}
if(id==3){ printf("\n\t Fail to open file to write\n");}
if(id==0){ printf("\n\t Results saved in CB_out.txt");}
printf("\n\n\t Press any key to quit.. \n");
getch();
exit(EXIT_FAILURE);}
/***********************************************/
/*============= 4. output() ==================*/
/***********************************************/
int output(){
int i, j;
outfp=fopen("CAB_out.txt","w");
if(outfp==NULL) return 3;
fprintf(outfp,"\n\n\t\t\tSimulation Settings");
fprintf(outfp,"\n");
fprintf(outfp,"\nMu            ");
for(j=1;j<=NS;j++) fprintf(outfp,"  & $%4.2f$",Mu[j]);
```

```
fprintf(outfp,"\nCommon sigma    & $%6.2f$",sigma);
fprintf(outfp,"\n1st percentile ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",P01[j]);
fprintf(outfp,"\n10-percentile ");
for(j=1;j<=NS;j++) fprintf(outfp,"  & $%4.2f$",P10[j]);
fprintf(outfp,"\n\n\n\t\t\tCC for A-basis\n");
fprintf(outfp,"\nANOVA          ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CA[0][j]);
fprintf(outfp,"\n\nAverage Size");
fprintf(outfp,"\nA-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CA[1][j]);
fprintf(outfp,"\nG-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CA[2][j]);
fprintf(outfp,"\nH-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CA[3][j]);
fprintf(outfp,"\nP-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CA[4][j]);
fprintf(outfp,"\n\n\n\t\t\tCC for B-basis\n");
fprintf(outfp,"\nANOVA           ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CB[0][j]);
fprintf(outfp,"\n\nAverage Size");
fprintf(outfp,"\nA-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CB[1][j]);
fprintf(outfp,"\nG-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CB[2][j]);
fprintf(outfp,"\nH-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CB[3][j]);
fprintf(outfp,"\nP-average      ");
for(j=1;j<=NS;j++) fprintf(outfp," & $%4.2f$",CB[4][j]);
fclose(outfp);
return 0;}
/*********************************************************/
/*============ main()=========*/
/*********************************************************/
void main(void){
clrscr();
int i,j,k,n,id;
double x, dAN, dGN, dHN, dPN;
/*-----------------------------------------
  (1) open file, read NS, u[], sigma, close file
-----------------------------------------------*/
infp=fopen("CAB_in.txt","r"); if(infp==NULL) close_down(1);
fscanf(infp,"%d",&NS);
for(i=1;i<=NS;i++) fscanf(infp,"%lf",&Mu[i]);
fscanf(infp,"%lf",&sigma);
```

```
fclose(infp);
/*-----------------------------------------
  (2) Create 1st and 10th percentiles in P01[], P10[]
      initialize CA[][], CB[][]
-----------------------------------------------*/
for(i=1;i<=NS;i++){
P01[i]=Mu[i]-2.32635*sigma; P10[i]=Mu[i]-1.28155*sigma;}
for(i=0;i<=4;i++)for(j=0;j<=NS;j++){
CA[i][j]=0; CB[i][j]=0;}
/*-----------------------------------------
   (3) i: 0: ANOVA: 1: A-average 2: G-average
          3: H-average; 4: P-average
       round: 1 to 5000
       show i and round on screen
-----------------------------------------------*/
randomize();
clrscr();
printf("\n\ti from 0 to 4; Round from 1 to 5000");
for(i=0;i<=4;i++){
for(Round=1;Round<=5000; Round++){
gotoxy(10,5); printf("i=%1d, Round=%5.0f",i, Round);
/*-----------------------------------------
 (3.1) For this i and this round
       fill random Size[] 5 to 49, Xbar[], Stdev[]
-----------------------------------------------*/
for(j=1;j<=NS;j++){
n=rand()%50; if(n<5) n=5; Size[j]=n;
simul_stat(n,Mu[j],sigma,&Xbar[j],&Stdev[j]); }
/*-----------------------------------------
  (3.2) For this i and this round get
        f and s for all, and d[] for
        for A-average, G-average, H-average, P-average
-----------------------------------------------*/
s=0; dAN=0; dGN=1; dHN=0; dPN=0;
for(j=1;j<=NS;j++){
x=(double) Size[j]; s=s+Stdev[j]*Stdev[j]*(x-1);
dAN+=x; dGN*=x; x=1.0/x; dHN+=x; dPN+=(x*x);}
x=(double)NS; f=dAN-x; s=sqrt(s/f); dAN/=x; dHN=x/dHN;
dPN=x/dPN; dPN=sqrt(dPN);
x=1.0/x; dGN=pow(dGN,x);
d[1]=dAN; d[2]=dGN; d[3]=dHN; d[4]=dPN;
/*-----------------------------------------
  (3.3) for this i and this round
        compute A-basis and B-basis (according to i)
        compare with percentiles
```

```
        record if it is a good one
-----------------------------------------------*/
if(i==0){
for(j=1;j<=NS;j++){
A=Xbar[j]-kA((double)Size[j],f)*s;
B=Xbar[j]-kB((double)Size[j],f)*s;
if(A<P01[j]) CA[i][j]++;
if(B<P10[j]) CB[i][j]++;}}
else if((i==1) ||(i==2)||(i==3)||(i==4)){
for(j=1;j<=NS;j++){
A=Xbar[j]-kA(d[i],f)*s;
B=Xbar[j]-kB(d[i],f)*s;
if(A<P01[j]) CA[i][j]++;
if(B<P10[j]) CB[i][j]++;}}
else close_down(2);}}
/*-------------------------------------------------
  (4) After all i, create estimated c.c.
      and call output();
-------------------------------------------------*/
for(i=0;i<=4;i++){
for(j=1;j<=NS;j++){
CA[i][j]/=5000;
CB[i][j]/=5000;}}
id=output();
if(id!=0) close_down(3);
close_down(0);}
```

OP Code

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAXN 11
/*--------input----------*/
FILE *infp, *outfp;
int VS;
double f[MAXN], w[MAXN];
int NP, O[MAXN][MAXN];
/*-------output---------*/
double p[MAXN];
/*****************************************************/
/*===============1. close_down() ====================*/
/*****************************************************/
void close_down(int id){
printf("\n\n");
if(id==1){
printf("\n\t Fail to open data file\n");
printf("\n\t Data file name: OP_in.txt\n");
printf("\n\t It contains");
printf("\n\t (1) VS --- vector size");
printf("\n\t (2) f[1]..f[VS] - vector f");
printf("\n\t (3) w[1]..w[VS] - vector w");
printf("\n\t (4) NP --- number of order pairs");
printf("\n\t (5) i1, j1..iNP,jNP - such that ik<=jk, k=1,...,NP");}
if(id==2){ printf("\n\t Fail to produce the projection");}
if(id==3){ printf("\n\t Fail to open file to write\n");}
if(id==0){ printf("\n\t Results saved in OP_out.txt");}
printf("\n\n\t Press any key to quit..\n");
getch();
exit(EXIT_FAILURE);}
/*****************************************************/
/*==============interface function==================*/
/*****************************************************/
int O_MC();
void main(void){
int i, j, k;
clrscr();
infp=fopen("OP_in.txt","r"); if(infp==NULL) close_down(1);
fscanf(infp,"%d",&VS);
for(i=1;i<=VS;i++) fscanf(infp,"%lf",&f[i]);
for(i=1;i<=VS;i++) fscanf(infp,"%lf",&w[i]);
```

```
fscanf(infp,"%d",&NP);
for(i=1;i<=VS;i++) for(j=1;j<=VS;j++) O[i][j]=0;
for(k=1;k<=NP;k++){
fscanf(infp,"%d",&i); fscanf(infp,"%d",&j);
O[i][j]=1;}
fclose(infp);
k=O_MC();
if(k!=0) close_down(2);
outfp=fopen("OP_out.txt","w");
if(outfp==NULL) close_down(3);
fprintf(outfp,"\n\n\tf     ");
for(i=1;i<=VS;i++) fprintf(outfp,"%9.4f",f[i]);
fprintf(outfp,"\n\n\tw     ");
for(i=1;i<=VS;i++) fprintf(outfp,"%9.4f",w[i]);
fprintf(outfp,"\n\n\tOrders");
for(i=1;i<=VS;i++) for(j=1;j<=VS;j++){
if(O[i][j]==1) fprintf(outfp,"  %1d<=%1d  ",i,j);}
fprintf(outfp,"\n\n\tp    ");
for(i=1;i<=VS;i++) fprintf(outfp,"%9.4f",p[i]);
fclose(outfp);
close_down(0);}
/*************************************************************
  int mc_process() takes
              int VectorLength
              double VectorF[]
              double VectorW[]
              int MatrixBO[][]
  produces    ResultVector[]

  Suppose a.cpp, a vector f[] of dim n is to be projected to
      order restriction cone with order defined in O[][] with
      respect to the metric defined by weights w[]. The projection
      will be in p[].
     (1) in a.cpp declare global variables
             int n;
             double f[];
             double w[];
             double int O[][];
             double p[];
     (2) Declare a function, say int O_MC(). This function is supposed to
             to produce the projection in p[]. So call this function when
             needed.
     (3) in this file, OP_MC.cpp, declare
            extern int n;
            extern double f[];
```

```
            extern double w[];
            extern int O[][];
            extern double p[];
         so these variables in a.cpp can be accessed from OP_MC.cpp
     (4) Define function int O_MC() in OP_MC.cpp. This function will
            Assign the values of n, f[], w[], O[][] to
                    VectorLength, VectorF[], VectorW[], MatrixBO[][]
            call O_MC()
            assign  ResultVector[] to p[]
            Return a value to show if it is a success
*************************************************************/
#include<stdio.h>
#include<conio.h>
#define MAXN    25
/*===== (1) Variables to be inittialized
    and to be taken            ====*/
int VectorLength;
double VectorF[MAXN], VectorW[MAXN];
int MatrixBO[MAXN][MAXN];
double ResultVector[MAXN];
/*===== (2) local tool variables ====== */
int MatrixIforJ[MAXN][MAXN], NodeState[MAXN], BaseI, TargetJ;

/** for NodeState **/
#define MERGED         0
#define UNPROCESSED    1
#define CHOPPED        2

/** for MatrixIforJ, MatrixBO **/
#define YES            1
#define NO             0
/*****************************************************
  0 Small tool functions
 *****************************************************/
/*==============================================
  01. n_1_state() -- # of 1 in state[]
 ==============================================*/
int number_of_unprocessed_node(void){
   int n_1=0; int i;
   for(i=1;i<=VectorLength;i++){
       if(NodeState[i]==UNPROCESSED) n_1++;}
   return n_1;}
/*==============================================
  02. Find BaseI
ResultVector has the largest value at BaseI
```

```
ResultVector has smallest value at BaseI
      Comments: Use only if there are unprocessed nodes
  ==========================================*/
void NodeIwithLargestValueinP(){
   double LargestValueinResultVector; int i,j;
   for(i=1;i<=VectorLength; i++){
      if(NodeState[i]==UNPROCESSED) {
BaseI=i; i=VectorLength+1; } }
   LargestValueinResultVector=ResultVector[BaseI];
   for(i=1;i<=VectorLength;i++){
     if(ResultVector[i]>LargestValueinResultVector &&
  NodeState[i]==UNPROCESSED) {
BaseI=i; LargestValueinResultVector=ResultVector[i];}}}
void NodeIwithSmallestValueinP(){
   double SmallestValueinResultVector; int i,j;
   for(i=1;i<=VectorLength; i++){
      if(NodeState[i]==UNPROCESSED) {
BaseI=i; i=VectorLength+1; } }
   SmallestValueinResultVector=ResultVector[BaseI];
   for(i=1;i<=VectorLength;i++){
     if(ResultVector[i]<SmallestValueinResultVector &&
  NodeState[i]==UNPROCESSED) {
BaseI=i; SmallestValueinResultVector=ResultVector[i];}}}
/*==========================================
  03. Target node J
    int up_J() returns # of upper path from I
      when it is 1, put target node in J
    int down_J() returns # of lower path from I
      when it is 1, put  target node in J
 ==========================================*/
int FindJ_fromI_up(void){
   int i, number_of_upper_path;   number_of_upper_path=0;
   for(i=1;i<=VectorLength;i++){
if(MatrixBO[BaseI][i]==YES) number_of_upper_path++;}
   if(number_of_upper_path==YES){
       for(i=1;i<=VectorLength;i++){
  if(MatrixBO[BaseI][i]==YES){ TargetJ=i; i=VectorLength+1; }}}
   return number_of_upper_path; }
int FindJ_fromI_down(void){
    int i, number_of_lower_path;   number_of_lower_path=0;
    for(i=1;i<=VectorLength;i++){
    if(MatrixBO[i][BaseI]==YES) number_of_lower_path++;}
    if(number_of_lower_path==YES){
       for(i=1;i<=VectorLength;i++){
  if(MatrixBO[i][BaseI]==YES){ TargetJ=i; i=VectorLength+1; }}}
```

```
    return number_of_lower_path; }
/*=============================================
  04 merge TargetJ is now a representative
  =============================================*/
void merge_LSP(void){
   int i, j;
   double r1, r2;
/*-----------------------------------------*/
/* Update Nodes represented by I, are now by J */
/*        I represents no one                */
/*-----------------------------------------*/
   for(i=1;i<=VectorLength;i++){
     if(MatrixIforJ[BaseI][i]==YES){
  MatrixIforJ[TargetJ][i]=YES;
  MatrixIforJ[BaseI][i]=NO;}}
/*-----------------------------------------**/
/* Update S:   state[I]=0 <===> I is merged  */
/**----------------------------------------***/
   NodeState[BaseI]=MERGED;
/*-----------------------------------------**/
/* Update p                                  */
/**----------------------------------------**/
   r1=0; r2=0;
   for(i=1;i<=VectorLength;i++){
if(MatrixIforJ[TargetJ][i]==YES){
    r1+=(VectorF[i]*VectorW[i]);
    r2+=(VectorW[i]);   }}
   r1=r1/r2;
   for(i=1;i<=VectorLength;i++){
      if(MatrixIforJ[TargetJ][i]==YES) ResultVector[i]=r1;} }
void up_merge(void){
   int i; double r1, r2;
 /*-----------------------------------*/
 /* Update Bo: convert i<=I to i<=J
      erase path to and from I */
 /**---------------------------------**/
   for(i=1;i<=VectorLength;i++){
      if(MatrixBO[i][BaseI]==YES) MatrixBO[i][TargetJ]=YES;}
   MatrixBO[TargetJ][TargetJ]=NO;
   for(i=1;i<=VectorLength;i++){
      MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   merge_LSP(); }
void down_merge(void){
   int i; double r1, r2;
/**-------------------------------------**/
```

```
/* Bo: Convert I<=i to J<=i, erase to, from I*/
/*-------------------------------------*****/
   for(i=1;i<=VectorLength;i++){
       if(MatrixBO[BaseI][i]==YES) MatrixBO[TargetJ][i]=YES;}
   MatrixBO[TargetJ][TargetJ]=NO;
   for(i=1;i<=VectorLength;i++){
       MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   merge_LSP(); }
/*==========================================
   05 chop off node I: update Bo, s
 ==========================================*/
void chop(void){
   int i;
   for(i=1;i<=VectorLength;i++) {
      MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   NodeState[BaseI]=CHOPPED;  }
/*****************************************/
/*  1. int mc_process() in main             */
/*****************************************/
int mc_process(void){
  int i, j, n1, n2, up_failure, down_failure;
  /*--------- Initialize L, s, p ------------*/
   for(i=1;i<=VectorLength;i++){
       for(j=1;j<=VectorLength;j++){ MatrixIforJ[i][j]=NO; } }
   for(i=1;i<=VectorLength;i++){
      MatrixIforJ[i][i]=YES;
      NodeState[i]=UNPROCESSED;
      ResultVector[i]=VectorF[i]; }
  while(1){
     up_failure=0; down_failure=0;
     n1=number_of_unprocessed_node(); if(n1==0) return 0;  /*Success */
     /*--- up process ---*/
     NodeIwithLargestValueinP();
     n2=FindJ_fromI_up();
     if(n2==0) chop();
        else if(n2==1) up_merge();
           else up_failure=1;
     n1=number_of_unprocessed_node(); if(n1==0) return 0;  /*Success */
     /*--- down process ---*/
     NodeIwithSmallestValueinP();
     n2=FindJ_fromI_down();
     if(n2==0) chop();
       else if(n2==1) down_merge();
            else down_failure=1;
     if(up_failure==1 && down_failure==1)
```

```
            return 2;                    /*Failure */ }      }
/*********************************************************/

/*******************************************************/
/*   interface function called in main()
       (1) Prepare int VectorLength
   double VectorF[], VectorW[]
   int MatrixBO[][]
       (2) call id=mc_process()
if(id!=0) return id;
       (3) Assign  ResultVector[] to outside vector
return 0
     to be called by other functions in other files     */
/*******************************************************/
extern int VS;
extern double f[11], w[11];
extern int O[11][11];
extern double p[11];
int O_MC(void){
   int i,j;
   /*---------------------------------
     prepare int    VectorLength
     double VectorF[]
     double VectorW[]
     int    MatrixBO[][]
     double ResultVector
     --------------------------------*/
     VectorLength=VS;
     for(i=1;i<=VectorLength;i++){
VectorF[i]=f[i];
VectorW[i]=w[i];}
     for(i=1;i<=VectorLength;i++)
for(j=1;j<=VectorLength;j++)
    MatrixBO[i][j]=O[i][j];
   /*-------------------------------
      call mc_process
     ------------------------------- */
     i=mc_process();
              if(i!=0) return i;
   /*-------------------------------
     take results
     -------------------------------*/
   for(i=1;i<=VectorLength;i++)
       p[i]=ResultVector[i];
   return 0; }
```

AB3 Code

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAXN 25
FILE *infp, *outfp;
int NS, Size[MAXN], NP, O[MAXN][MAXN];
double Xbar[MAXN], Stdev[MAXN], Omean[MAXN];
double HN, f, s;
double A[3][MAXN], B[3][MAXN];
/*********************************************/
/*         1. kA(d,f) and kB(d,f)            */
/*********************************************/
double p4(double a0, double a1, double a2, double a3, double a4, double x){
double r; r=a3+a4*x;r=a2+r*x;r=a1+r*x;r=a0+r*x; return r; }
double myK(double z, double q, double d, double b, double c){
double r1,r2,r3; r1=z/sqrt(q); r3=0.5*b/c; r2=1.0/d;
r2=r2/c; r2=r2+r3*r3; r2=sqrt(r2); r1=r1+r2-r3; return r1; }
double kA(double d, double f){
double x,q,b,c,K;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,0.6057,-0.3287,x);
b=p4(0,2.0643,-0.95145,0.51251,0,x);
c=p4(0.36961,0.0026958,-0.65201,0.011320,0,x);
K=myK(2.32635,q,d,b,c);
return K;}
double kB(double d, double f){
double x, q,b,c,K;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,0.6057,-0.3287,x);
b=p4(0,1.1372,-0.49162,0.18612,0,x);
c=p4(0.36961,0.0040342,-0.71750,0.19693,0,x);
K=myK(1.28155,q,d,b,c);
return K; }
/*********************************************/
/*            2. close_down                  */
/*********************************************/
void close_down(int id){
if(id==1){
printf("\n\t Fail to open file to read!\n");
printf("\n\t Input file name: AB3_in.txt\n");
printf("\n\t It contains");
printf("\n\t  (1) NS - Number of samples");
```

```
printf("\n\t  (2) Size[1]..Size[NS] - Sample sizes");
printf("\n\t  (3) sample1...sampleNS - Samples");
printf("\n\t  (4) NP - Number of Order pairs");
printf("\n\t  (5) ik, jk, k=1...NP - ik<=jk imposed"); }
if(id==2){ printf("\n\t Fail to produce RMLE of means");}
if(id==3){ printf("\n\t Fail to open file to write"); }
if(id==0){ printf("\n\t Results saved in AB3_out.txt");}
printf("\n\n\t Press any key to quit\n");
getch();
exit(0);}
/*********************************************/
/*         3. int AB3_OMC()                  */
/*********************************************/
int AB3_OMC(void);
/*===========================================*/
void main(void){
clrscr();
int i,j,k; double x,sx,sxx;
/*----------------------------------
  open file, read NS, Size[], samples
  compute Xbar[], Stdev[]
  read NP, O[], close file
  ----------------------------------*/
infp=fopen("AB3_in.txt","r");
if(infp==NULL) close_down(1);
fscanf(infp,"%d",&NS);
for(i=1;i<=NS;i++) fscanf(infp,"%d",&Size[i]);
for(i=1;i<=NS;i++){ sx=0; sxx=0;
for(j=1;j<=Size[i];j++){
fscanf(infp,"%lf",&x); sx+=x; sxx+=(x*x);}
x=(double) Size[i]; Xbar[i]=sx/x;
sxx=sxx-sx*sx/x; sxx=sxx/(x-1);
Stdev[i]=sqrt(sxx);}
fscanf(infp,"%d",&NP);
for(i=1;i<=NS;i++) for(j=1;j<=NS;j++)
O[i][j]=0;
for(k=1;k<=NP;k++){
fscanf(infp,"%d",&i); fscanf(infp,"%d",&j);
O[i][j]=1;}
fclose(infp);
/*----------------------------------
  compute f, s, HN
  ----------------------------------*/
f=0;s=0;HN=0;
for(i=1;i<=NS;i++){
```

```
x=(double) Size[i]; f+=x; s=s+Stdev[i]*Stdev[i]*(x-1);
x=1/x; HN+=x;}
x=(double) NS; f-=x; s=sqrt(s/f);
HN=x/HN;
/*--------------------------------
  Produce Omean[]
  --------------------------------*/
k=AB3_OMC(); if(k!=0) close_down(2);
/*--------------------------------------
  KS: first row and first column of A, B
  --------------------------------------*/
for(j=1;j<=NS;j++){
A[0][j]=Xbar[j]-kA((double)Size[j],f)*s;
B[0][j]=Xbar[j]-kB((double)Size[j],f)*s;
A[1][j]=Xbar[j]-kA(HN,f)*s;
B[1][j]=Xbar[j]-kB(HN,f)*s;
A[2][j]=Omean[j]-kA(HN,f)*s;
B[2][j]=Omean[j]-kB(HN,f)*s; }
/*---------------------------------
  output
  ---------------------------------*/
outfp=fopen("AB3_out.txt","w");
if(outfp==NULL) close_down(3);
fprintf(outfp,"\n\n\t\tA-basis and B-basis\n");
fprintf(outfp,"\n\tS-Size   ");
for(i=1;i<=NS;i++){
fprintf(outfp,"  %5d   ",Size[i]);}
fprintf(outfp,"\n\tS-Stdev  ");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",Stdev[i]);}
fprintf(outfp,"\n\tXbar     ");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",Xbar[i]);}
fprintf(outfp,"\n\tRMLE     ");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",Omean[i]);}
fprintf(outfp,"\n\n");
fprintf(outfp,"\n\tA-basis");
fprintf(outfp,"\n\tANOVA     ");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",A[0][i]);}
fprintf(outfp,"\n\tH-average ");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",A[1][i]);}
fprintf(outfp,"\n\tRMLE H-ave");
```

```
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",A[2][i]);}
fprintf(outfp,"\n\n");
fprintf(outfp,"\n\tB-basis");
fprintf(outfp,"\n\tANOVA     ");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",B[0][i]);}
fprintf(outfp,"\n\tH-average ");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",B[1][i]);}
fprintf(outfp,"\n\tRMLE H-ave");
for(i=1;i<=NS;i++){
fprintf(outfp," %9.4f",B[2][i]);}
fprintf(outfp,"\n\n");
fprintf(outfp,"\n\tOrdering Imposed: ");
for(i=1;i<=NS;i++) for(j=1;j<=NS;j++){
if(O[i][j]==1) fprintf(outfp," %2d<=%2d",i,j);}
fclose(outfp);
close_down(0);}
/*************************************************************
  int mc_process() takes
              int VectorLength
              double VectorF[]
              double VectorW[]
              int MatrixBO[][]
  produces    ResultVector[]

  Suppose a.cpp, a vector f[] of dim n is to be projected to
      order restriction cone with order defined in O[][] with
      respect to the metric defined by weights w[]. The projection
      will be in p[].
     (1) in a.cpp declare global variables
             int n;
             double f[];
             double w[];
             double int O[][];
             double p[];
     (2) Declare a function, say int O_MC(). This function is supposed to
             to produce the projection in p[]. So call this function when
             needed.
     (3) in this file, OP_MC.cpp, declare
            extern int n;
            extern double f[];
            extern double w[];
            extern int O[][];
```

```
            extern double p[];
         so these variables in a.cpp can be accessed from OP_MC.cpp
     (4) Define function int O_MC() in OP_MC.cpp. This function will
            Assign the values of n, f[], w[], O[][] to
                    VectorLength, VectorF[], VectorW[], MatrixBO[][]
            call O_MC()
            assign  ResultVector[] to p[]
            Return a value to show if it is a success
************************************************************/
#include<stdio.h>
#include<conio.h>
#define MAXN    25
/*===== (1) Variables to be initialized
    and to be taken            ====*/
int VectorLength;
double VectorF[MAXN], VectorW[MAXN];
int MatrixBO[MAXN][MAXN];
double ResultVector[MAXN];
/*===== (2) local tool variables ====== */
int MatrixIforJ[MAXN][MAXN], NodeState[MAXN], BaseI, TargetJ;
/** for NodeState **/
#define MERGED         0
#define UNPROCESSED    1
#define CHOPPED        2
/** for MatrixIforJ, MatrixBO **/
#define YES            1
#define NO             0
/*********************************************************
  0 Small tool functions
 *********************************************************/
/*===============================================
  01. n_1_state() -- # of 1 in state[]
 ===============================================*/
int number_of_unprocessed_node(void){
   int n_1=0; int i;
   for(i=1;i<=VectorLength;i++){
       if(NodeState[i]==UNPROCESSED) n_1++;}
   return n_1;}
/*===============================================
  02. Find BaseI
ResultVector has the largest value at BaseI
ResultVaector has smallest value at BaseI
      Comments: Use only if there are unprocessed nodes
  ===============================================*/
void NodeIwithLargestValueinP(){
```

```
   double LargestValueinResultVector; int i,j;
   for(i=1;i<=VectorLength; i++){
      if(NodeState[i]==UNPROCESSED) {
BaseI=i; i=VectorLength+1; } }
   LargestValueinResultVector=ResultVector[BaseI];
   for(i=1;i<=VectorLength;i++){
     if(ResultVector[i]>LargestValueinResultVector &&
  NodeState[i]==UNPROCESSED) {
BaseI=i; LargestValueinResultVector=ResultVector[i];}}}
void NodeIwithSmallestValueinP(){
   double SmallestValueinResultVector; int i,j;
   for(i=1;i<=VectorLength; i++){
      if(NodeState[i]==UNPROCESSED) {
BaseI=i; i=VectorLength+1; } }
   SmallestValueinResultVector=ResultVector[BaseI];
   for(i=1;i<=VectorLength;i++){
     if(ResultVector[i]<SmallestValueinResultVector &&
  NodeState[i]==UNPROCESSED) {
BaseI=i; SmallestValueinResultVector=ResultVector[i];}}}
/*=============================================
  03. Target node J
    int up_J() returns # of upper path from I
      when it is 1, put target node in J
    int down_J() returns # of lower path from I
      when itis 1, put  target node in J
 =============================================*/
int FindJ_fromI_up(void){
   int i, number_of_upper_path;   number_of_upper_path=0;
   for(i=1;i<=VectorLength;i++){
if(MatrixBO[BaseI][i]==YES) number_of_upper_path++;}
   if(number_of_upper_path==YES){
       for(i=1;i<=VectorLength;i++){
  if(MatrixBO[BaseI][i]==YES){ TargetJ=i; i=VectorLength+1; }}}
   return number_of_upper_path; }
int FindJ_fromI_down(void){
    int i, number_of_lower_path;   number_of_lower_path=0;
    for(i=1;i<=VectorLength;i++){
    if(MatrixBO[i][BaseI]==YES) number_of_lower_path++;}
    if(number_of_lower_path==YES){
       for(i=1;i<=VectorLength;i++){
  if(MatrixBO[i][BaseI]==YES){ TargetJ=i; i=VectorLength+1; }}}
    return number_of_lower_path; }
/*=============================================
  04 merge TargetJ is now a representative
  =============================================*/
```

```
void merge_LSP(void){
   int i, j;
   double r1, r2;
/*-----------------------------------------*/
/* Update Nodes represented by I, are now by J */
/*        I represents no one                */
/*-----------------------------------------*/
   for(i=1;i<=VectorLength;i++){
     if(MatrixIforJ[BaseI][i]==YES){
  MatrixIforJ[TargetJ][i]=YES;
  MatrixIforJ[BaseI][i]=NO;}}
/*-----------------------------------------**/
/* Update S:   state[I]=0 <===> I is merged  */
/**----------------------------------------***/
   NodeState[BaseI]=MERGED;
/*-----------------------------------------**/
/* Update p                                  */
/**----------------------------------------**/
   r1=0; r2=0;
   for(i=1;i<=VectorLength;i++){
if(MatrixIforJ[TargetJ][i]==YES){
    r1+=(VectorF[i]*VectorW[i]);
    r2+=(VectorW[i]);   }}
   r1=r1/r2;
   for(i=1;i<=VectorLength;i++){
      if(MatrixIforJ[TargetJ][i]==YES) ResultVector[i]=r1;} }
void up_merge(void){
   int i; double r1, r2;
 /*-----------------------------------*/
 /* Update Bo: convert i<=I to i<=J
      erase path to and from I */
 /**----------------------------------**/
   for(i=1;i<=VectorLength;i++){
      if(MatrixBO[i][BaseI]==YES) MatrixBO[i][TargetJ]=YES;}
   MatrixBO[TargetJ][TargetJ]=NO;
   for(i=1;i<=VectorLength;i++){
      MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   merge_LSP(); }
void down_merge(void){
   int i; double r1, r2;
/**---------------------------------------**/
/* Bo: Convert I<=i to J<=i, erase to, from I*/
/*---------------------------------------*****/
   for(i=1;i<=VectorLength;i++){
       if(MatrixBO[BaseI][i]==YES) MatrixBO[TargetJ][i]=YES;}
```

```
   MatrixBO[TargetJ][TargetJ]=NO;
   for(i=1;i<=VectorLength;i++){
       MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   merge_LSP(); }
/*=========================================
   05 chop off node I: update Bo, s
 =========================================*/
void chop(void){
   int i;
   for(i=1;i<=VectorLength;i++) {
      MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   NodeState[BaseI]=CHOPPED;  }
/*****************************************/
/*  1. int mc_process() in main             */
/*****************************************/
int mc_process(void){
  int i, j, n1, n2, up_failure, down_failure;
  /*--------- Initialize L, s, p ------------*/
   for(i=1;i<=VectorLength;i++){
       for(j=1;j<=VectorLength;j++){ MatrixIforJ[i][j]=NO; } }
   for(i=1;i<=VectorLength;i++){
      MatrixIforJ[i][i]=YES;
      NodeState[i]=UNPROCESSED;
      ResultVector[i]=VectorF[i]; }
  while(1){
     up_failure=0; down_failure=0;
     n1=number_of_unprocessed_node(); if(n1==0) return 0;  /*Success */
     /*--- up process ---*/
     NodeIwithLargestValueinP();
     n2=FindJ_fromI_up();
     if(n2==0) chop();
        else if(n2==1) up_merge();
           else up_failure=1;
     n1=number_of_unprocessed_node(); if(n1==0) return 0;  /*Success */
     /*--- down process ---*/
     NodeIwithSmallestValueinP();
     n2=FindJ_fromI_down();
     if(n2==0) chop();
       else if(n2==1) down_merge();
            else down_failure=1;
     if(up_failure==1 && down_failure==1)
            return 2;                    /*Failure */     }      }
/*********************************************************/

/*******************************************************/
```

```
/*   interface function called in main()
       (1) Prepare int VectorLength
   double VectorF[], VectorW[]
   int MatrixBO[][]
       (2) call id=mc_process()
if(id!=0) return id;
       (3) Assign  ResultVector[] to outside vector
return 0
     to be called by other functions in other files     */
/*******************************************************/
extern int NS;
extern int Size[25];
extern double Xbar[25];
extern int O[25][25];
extern double Omean[25];
int AB3_OMC(void){
   int i,j;
   /*--------------------------------
     prepare int    VectorLength
     double VectorF[]
     double VectorW[]
     int    MatrixBO[][]
     double ResultVector
     --------------------------------*/
     VectorLength=NS;
     for(i=1;i<=VectorLength;i++){
VectorF[i]=Xbar[i];
VectorW[i]=(double) Size[i];}
     for(i=1;i<=VectorLength;i++)
for(j=1;j<=VectorLength;j++)
    MatrixBO[i][j]=O[i][j];
   /*-------------------------------
      call mc_process
     ------------------------------ */
     i=mc_process();
              if(i!=0) return i;
   /*-------------------------------
     take results
     ------------------------------*/
   for(i=1;i<=VectorLength;i++){
       Omean[i]=ResultVector[i];}
   return 0; }
```

FINAL Code

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAXN 25
/*----- input -----*/
FILE *infp; FILE *outfp;
int NS, NP, O[MAXN][MAXN];
double Mu[MAXN], sigma;
/*---- computation ---*/
double P10[MAXN], P01[MAXN];
double CA[3][MAXN], CB[3][MAXN];
double Round;
int Size[MAXN];
double Xbar[MAXN], Stdev[MAXN], A, B,C,D;
double f, s;
double d[MAXN], p[MAXN];
/*****************************************/
/********** 1. simul_stat() **********/
/*****************************************/
double obsN(double mu, double sigma){
int i, n=40; double r=0.0;
for(i=0;i<n;i++) r+=(rand()%100);
r-=(49.5*n); r/=sqrt(3333.0*n/4);
r*=sigma; r+=mu; return r; }
void simul_stat(int n, double mu, double sigma, double *Sm, double *Ss){
int i; double x, sx, sxx; sx=0; sxx=0;
for(i=1;i<=n;i++){x=obsN(mu,sigma); sx+=x; sxx+=(x*x);}
x=sx/(double)n; *Sm=x;
sx=sx*sx/(double)n; sxx=sxx-sx; sxx=sxx/(double)(n-1);
sxx=sqrt(sxx); *Ss=sxx;}
/*****************************************/
/******* 2. kA(d,f), kB(d,f) *********/
/*****************************************/
double p4(double a0, double a1, double a2, double a3, double a4, double x){
double r;
r=a3+a4*x; r=a2+r*x; r=a1+r*x; r=a0+r*x;
return r;}
double k(double z, double q, double d, double b, double c){
double r1, r2, r3;
r1=z/sqrt(q); r3=0.5*b/c; r2=1.0/d;
r2=r2/c; r2=r2+r3*r3; r2=sqrt(r2); r1=r1+r2-r3;
return r1;}
```

```
double kA(double d, double f){
double x,q,b,c, k_A;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,0.6057,-0.3287,x);
b=p4(0,2.0643,-0.95145,0.51251,0,x);
c=p4(0.36961,0.0026958,-0.65201,0.011320,0,x);
k_A=k(2.32635,q,d,b,c);
return k_A;}
double kB(double d, double f){
double x, q,b,c, k_B;
x=sqrt(1/f);
q=p4(1,-2.327,1.138,0.6057,-0.3287,x);
b=p4(0,1.1372,-0.49162,0.18612,0,x);
c=p4(0.36961,0.0040342,-0.71750,0.19693,0,x);
k_B=k(1.28155,q,d,b,c);
return k_B;}
/*********************************************/
/* ======== 3. close_down() ==================*/
/*********************************************/
void close_down(int id){
printf("\n\n");
if(id==1){
printf("\n\t Fail to open data file\n");
printf("\n\t Data file name: RAB-in.txt\n");
printf("\n\t It contains");
printf("\n\t (1) NS --- number of samples");
printf("\n\t (2) Mu[1]..Mu[NS] - population means");
printf("\n\t (3) sigma --- population stdev");
printf("\n\t (4) NP - number of order pairs");
printf("\n\t (5) ik, jk, k=1...NP - ik<=jk imposed");}
if(id==2){ printf("\n\t Case classification error");}
if(id==3){ printf("\n\t Fail to open file to write\n");}
if(id==0){ printf("\n\t Results saved in CAB_out.txt");}
if(id==4){ printf("\n\t Fail to produce RMLE of means");}
printf("\n\n\t Press any key to quit.. \n");
getch();
exit(EXIT_FAILURE);}
/*********************************************/
/*============= 4. output() ==================*/
/*********************************************/
int output(){
int i, j;
outfp=fopen("CAB_out.txt","w");
if(outfp==NULL) return 3;
fprintf(outfp,"\n\n\t\t\tSimulated Confidence Coefficients for A-basis and B-basis");
```

```
fprintf(outfp,"\n\n\ANOVA settings");
fprintf(outfp,"\n# of populations  & $%5d $",NS);
fprintf(outfp,"\nCommon Sigma      & $%6.2f$",sigma);
fprintf(outfp,"\nMeans             ");
for(j=1;j<=NS;j++)
fprintf(outfp,"& $%6.2f$ ",Mu[j]);
fprintf(outfp,"\n# of Rounds       & $  5000$");
fprintf(outfp,"\nRandom S-Size     & $  5-49$");
fprintf(outfp,"\n");
fprintf(outfp,"\nMu                ");
for(j=1;j<=NS;j++)
fprintf(outfp,"& $%6.2f$ ",Mu[j]);
fprintf(outfp,"\n1st percentile    ");
for(j=1;j<=NS;j++)
fprintf(outfp,"& $%6.2f$ ",P01[j]);
fprintf(outfp,"\n10-percentile     ");
for(j=1;j<=NS;j++)
fprintf(outfp,"& $%6.2f$ ",P10[j]);
fprintf(outfp,"\n\n\nCC for A-basis");
fprintf(outfp,"\nANOVA             ");
for(j=1;j<=NS;j++) fprintf(outfp,"& $%6.4f$ ",CA[0][j]);
fprintf(outfp,"\nH-average         ");
for(j=1;j<=NS;j++) fprintf(outfp,"& $%6.4f$ ",CA[1][j]);
fprintf(outfp,"\nRestricted        ");
for(j=1;j<=NS;j++) fprintf(outfp,"& $%6.4f$ ",CA[2][j]);
fprintf(outfp,"\n\n\nCC for B-basis");
fprintf(outfp,"\nANOVA             ");
for(j=1;j<=NS;j++) fprintf(outfp,"& $%6.4f$ ",CB[0][j]);
fprintf(outfp,"\nH-average         ");
for(j=1;j<=NS;j++) fprintf(outfp,"& $%6.4f$ ",CB[1][j]);
fprintf(outfp,"\nRestricted        ");
for(j=1;j<=NS;j++) fprintf(outfp,"& $%6.4f$ ",CB[2][j]);
fprintf(outfp,"\n\nRestrictions\n  ");
for(i=1;i<=NS;i++) for(j=1;j<=NS;j++){
if(O[i][j]==1) fprintf(outfp," %2d<=%2d",i,j);}
fclose(outfp);
return 0;}
/*********************************************************/
/*============ main()=========*/
/*********************************************************/
void main(void){
clrscr();
int i,j,k,n,id;
double x, dHN, dAN;
/*----------------------------------------
```

```
  (1) open file, read NS, Mu[], sigma, NP, close file
-----------------------------------------------*/
for(i=1;i<=NS;i++) for(j=1;j<=NS;j++){
O[i][j]=0;}
infp=fopen("CAB_in.txt","r"); if(infp==NULL) close_down(1);
fscanf(infp,"%d",&NS);
for(i=1;i<=NS;i++) fscanf(infp,"%lf",&Mu[i]);
fscanf(infp,"%lf",&sigma);
fscanf(infp,"%d",&NP);
for(k=1;k<=NP;k++){
fscanf(infp,"%d",&i);
fscanf(infp,"%d",&j);
O[i][j]=1;}
fclose(infp);
/*----------------------------------------
  (2) Create 1st and 10th percentiles in P01[], P10[]
      initialize CA[][], CB[][]
-----------------------------------------------*/
for(i=1;i<=NS;i++){
P01[i]=Mu[i]-2.32635*sigma; P10[i]=Mu[i]-1.28155*sigma;}
for(i=0;i<=2;i++)for(j=0;j<=NS;j++){
CA[i][j]=0; CB[i][j]=0;}
/*----------------------------------------
   (3) i: 0: ANOVA: 1: H-average; 2: Restricted
       round: 1 to 5000
       show i and round on screen
-----------------------------------------------*/
randomize();
clrscr();
printf("\n\ti from 0 to 1; Round from 1 to 5000");
for(i=0;i<=2;i++){
for(Round=1;Round<=5000;Round++){
gotoxy(10,5); printf("i=%1d, Round=%5.0f",i, Round);
/*----------------------------------------
 (3.1) For this i and this round
       fill random Size[] 5 to 49, Xbar[], Stdev[]
-----------------------------------------------*/
for(j=1;j<=NS;j++){
n=rand()%50; if(n<5) n=5; Size[j]=n;
simul_stat(n,Mu[j],sigma,&Xbar[j],&Stdev[j]); }
/*----------------------------------------
  (3.2) For this i and this round get
        f and s for all, and d[] for
        for H-average and Restricted
-----------------------------------------------*/
```

```
s=0; dHN=0; dAN=0;
for(j=1;j<=NS;j++){
x=(double) Size[j]; s=s+Stdev[j]*Stdev[j]*(x-1);
dAN+=x;
x=1.0/x; dHN+=x; }
x=(double)NS; f=dAN-x; s=sqrt(s/f); dHN=x/dHN;
x=1.0/x;
d[1]=dHN;
/*-----------------------------------------------
  (3.3) for this i and this round
        compute A-basis and B-basis (according to i)
        compare with percentiles
        record if it is a good one
-----------------------------------------------*/
if(i==0){
for(j=1;j<=NS;j++){
A=Xbar[j]-kA((double)Size[j],f)*s;
B=Xbar[j]-kB((double)Size[j],f)*s;
if(A<P01[j]) CA[i][j]++;
if(B<P10[j]) CB[i][j]++;}}
else if((i==1)) {
for(j=1;j<=NS;j++){
A=Xbar[j]-kA(d[i],f)*s;
B=Xbar[j]-kB(d[i],f)*s;
if(A<P01[j]) CA[i][j]++;
if(B<P10[j]) CB[i][j]++;}}
else if((i==2)){
int O_MC(void);
k=O_MC();
if(k!=0) close_down(4);
for(j=1;j<=NS;j++){
C=p[j]-kA(d[1],f)*s;
D=p[j]-kB(d[1],f)*s;
if(C<P01[j]) CA[2][j]++;
if(D<P10[j]) CB[2][j]++;}}
else close_down(2);}}
/*-------------------------------------------------
  (4) After all i, create estimated c.c.
      and call output();
-------------------------------------------------*/
int l;
for(l=0;l<=2;l++){
for(j=1;j<=NS;j++){
CA[l][j]/=5000;
CB[l][j]/=5000;}}
```

```
id=output();
if(id!=0) close_down(3);
close_down(0);}
/*************************************************************
  int mc_process() takes
              int VectorLength
              double VectorF[]
              double VectorW[]
              int MatrixBO[][]
  produces    ResultVector[]

  Suppose a.cpp, a vector f[] of dim n is to be projected to
      order restriction cone with order defined in O[][] with
      respect to the metric defined by weights w[]. The projection
      will be in p[].
     (1) in a.cpp declare global variables
             int n;
             double f[];
             double w[];
             double int O[][];
             double p[];
     (2) Declare a function, say int O_MC(). This function is supposed to
             to produce the projection in p[]. So call this function when
             needed.
     (3) in this file, OP_MC.cpp, declare
            extern int n;
            extern double f[];
            extern double w[];
            extern int O[][];
            extern double p[];
         so these variables in a.cpp can be accessed from OP_MC.cpp
     (4) Define function int O_MC() in OP_MC.cpp. This function will
            Assign the values of n, f[], w[], O[][] to
                    VectorLength, VectorF[], VectorW[], MatrixBO[][]
            call O_MC()
            assign  ResultVector[] to p[]
            Return a value to show if it is a success
*************************************************************/
#include<stdio.h>
#include<conio.h>
#define MAXN    25
/*===== (1) Variables to be initialized
    and to be taken            ====*/
int VectorLength;
double VectorF[MAXN], VectorW[MAXN];
```

```
int MatrixBO[MAXN][MAXN];
double ResultVector[MAXN];
/*===== (2) local tool variables ====== */
int MatrixIforJ[MAXN][MAXN], NodeState[MAXN], BaseI, TargetJ;
/** for NodeState **/
#define MERGED         0
#define UNPROCESSED    1
#define CHOPPED        2
/** for MatrixIforJ, MatrixBO **/
#define YES            1
#define NO             0
/*********************************************************
  0 Small tool functions
 *********************************************************/
/*===============================================
  01. n_1_state() -- # of 1 in state[]
 ================================================*/
int number_of_unprocessed_node(void){
   int n_1=0; int i;
   for(i=1;i<=VectorLength;i++){
       if(NodeState[i]==UNPROCESSED) n_1++;}
   return n_1;}
/*===============================================
  02. Find BaseI
ResultVector has the largest value at BaseI
ResultVector has smallest value at BaseI
      Comments: Use only if there are unprocessed nodes
  ================================================*/
void NodeIwithLargestValueinP(){
   double LargestValueinResultVector; int i,j;
   for(i=1;i<=VectorLength; i++){
      if(NodeState[i]==UNPROCESSED) {
BaseI=i; i=VectorLength+1; } }
   LargestValueinResultVector=ResultVector[BaseI];
   for(i=1;i<=VectorLength;i++){
     if(ResultVector[i]>LargestValueinResultVector &&
  NodeState[i]==UNPROCESSED) {
BaseI=i; LargestValueinResultVector=ResultVector[i];}}}
void NodeIwithSmallestValueinP(){
   double SmallestValueinResultVector; int i,j;
   for(i=1;i<=VectorLength; i++){
      if(NodeState[i]==UNPROCESSED) {
BaseI=i; i=VectorLength+1; } }
   SmallestValueinResultVector=ResultVector[BaseI];
   for(i=1;i<=VectorLength;i++){
```

```
     if(ResultVector[i]<SmallestValueinResultVector &&
  NodeState[i]==UNPROCESSED) {
BaseI=i; SmallestValueinResultVector=ResultVector[i];}}}
/*=============================================
  03. Target node J
    int up_J() returns # of upper path from I
      when it is 1, put target node in J
    int down_J() returns # of lower path from I
      when it is 1, put  target node in J
 =============================================*/
int FindJ_fromI_up(void){
   int i, number_of_upper_path;   number_of_upper_path=0;
   for(i=1;i<=VectorLength;i++){
if(MatrixBO[BaseI][i]==YES) number_of_upper_path++;}
   if(number_of_upper_path==YES){
       for(i=1;i<=VectorLength;i++){
  if(MatrixBO[BaseI][i]==YES){ TargetJ=i; i=VectorLength+1; }}}
   return number_of_upper_path; }
int FindJ_fromI_down(void){
    int i, number_of_lower_path;   number_of_lower_path=0;
    for(i=1;i<=VectorLength;i++){
    if(MatrixBO[i][BaseI]==YES) number_of_lower_path++;}
    if(number_of_lower_path==YES){
       for(i=1;i<=VectorLength;i++){
  if(MatrixBO[i][BaseI]==YES){ TargetJ=i; i=VectorLength+1; }}}
    return number_of_lower_path; }
/*=============================================
  04 merge TargetJ is now a representative
  =============================================*/
void merge_LSP(void){
   int i, j;
   double r1, r2;
/*-------------------------------------------*/
/* Update Nodes reprented by I, are now by J */
/*        I represents no one                */
/*-------------------------------------------*/
   for(i=1;i<=VectorLength;i++){
     if(MatrixIforJ[BaseI][i]==YES){
  MatrixIforJ[TargetJ][i]=YES;
  MatrixIforJ[BaseI][i]=NO;}}
/*-------------------------------------------**/
/* Update S:   state[I]=0 <===> I is merged  */
/**-----------------------------------------***/
   NodeState[BaseI]=MERGED;
/*-------------------------------------------**/
```

```
/* Update p                                      */
/**---------------------------------------**/
   r1=0; r2=0;
   for(i=1;i<=VectorLength;i++){
if(MatrixIforJ[TargetJ][i]==YES){
    r1+=(VectorF[i]*VectorW[i]);
    r2+=(VectorW[i]);   }}
   r1=r1/r2;
   for(i=1;i<=VectorLength;i++){
      if(MatrixIforJ[TargetJ][i]==YES) ResultVector[i]=r1;} }
void up_merge(void){
   int i; double r1, r2;
 /*-----------------------------------*/
 /* Update Bo: convert i<=I to i<=J
      erase path to and from I */
 /**---------------------------------**/
   for(i=1;i<=VectorLength;i++){
      if(MatrixBO[i][BaseI]==YES) MatrixBO[i][TargetJ]=YES;}
   MatrixBO[TargetJ][TargetJ]=NO;
   for(i=1;i<=VectorLength;i++){
      MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   merge_LSP(); }
void down_merge(void){
   int i; double r1, r2;
/**-------------------------------------**/
/* Bo: Convert I<=i to J<=i, erase to, from I*/
/*-------------------------------------*****/
   for(i=1;i<=VectorLength;i++){
       if(MatrixBO[BaseI][i]==YES) MatrixBO[TargetJ][i]=YES;}
   MatrixBO[TargetJ][TargetJ]=NO;
   for(i=1;i<=VectorLength;i++){
       MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   merge_LSP(); }
/*==========================================
   05 chop off node I: update Bo, s
 ==========================================*/
void chop(void){
   int i;
   for(i=1;i<=VectorLength;i++) {
      MatrixBO[i][BaseI]=NO; MatrixBO[BaseI][i]=NO;}
   NodeState[BaseI]=CHOPPED;  }
/*****************************************/
/*  1. int mc_process() in main             */
/*****************************************/
int mc_process(void){
```

```
  int i, j, n1, n2, up_failure, down_failure;
  /*--------- Initialize L, s, p ------------*/
   for(i=1;i<=VectorLength;i++){
       for(j=1;j<=VectorLength;j++){ MatrixIforJ[i][j]=NO; } }
   for(i=1;i<=VectorLength;i++){
      MatrixIforJ[i][i]=YES;
      NodeState[i]=UNPROCESSED;
      ResultVector[i]=VectorF[i]; }
  while(1){
     up_failure=0; down_failure=0;
     n1=number_of_unprocessed_node(); if(n1==0) return 0;  /*Success */
     /*--- up process ---*/
     NodeIwithLargestValueinP();
     n2=FindJ_fromI_up();
     if(n2==0) chop();
        else if(n2==1) up_merge();
           else up_failure=1;
     n1=number_of_unprocessed_node(); if(n1==0) return 0;  /*Success */
     /*--- down process ---*/
     NodeIwithSmallestValueinP();
     n2=FindJ_fromI_down();
     if(n2==0) chop();
       else if(n2==1) down_merge();
            else down_failure=1;
     if(up_failure==1 && down_failure==1)
            return 2;                    /*Failure */     }      }
/*********************************************************/

/*******************************************************/
/*   interface function called in main()
       (1) Prepare int VectorLength
   double VectorF[], VectorW[]
   int MatrixBO[][]
       (2) call id=mc_process()
if(id!=0) return id;
       (3) Assign  ResultVector[] to outside vector
return 0
     to be called by other functions in other files     */
/*******************************************************/
extern int NS, Size[25];
extern double Xbar[25];
extern int O[25][25];
extern double p[25];
int O_MC(void){
   int i,j;
```

```
   /*--------------------------------
     prepare int    VectorLength
     double VectorF[]
     double VectorW[]
     int    MatrixBO[][]
     double ResultVector
     --------------------------------*/
     VectorLength=NS;
     for(i=1;i<=VectorLength;i++){
VectorF[i]=Xbar[i];
VectorW[i]=(double) Size[i];}
     for(i=1;i<=VectorLength;i++)
for(j=1;j<=VectorLength;j++)
    MatrixBO[i][j]=O[i][j];
   /*-------------------------------
      call mc_process
     ------------------------------ */
     i=mc_process();
              if(i!=0) return i;
   /*-------------------------------
     take results
     ------------------------------*/
   for(i=1;i<=VectorLength;i++)
       p[i]=ResultVector[i];
   return 0; }
```