IMPLEMENTATION OF FIXED AND SEQUENTIAL MULTILEVEL ACCEPTANCE SAMPLING: THE R PACKAGE MFSAS

IMPLEMENTATION OF FIXED AND SEQUENTIAL MULTILEVEL ACCEPTANCE SAMPLING: THE R PACKAGE MFSAS

By

YALIN CHEN, B.Sc

A Project

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2010)	McMaster University
(Statistics)	Hamilton, Ontario

TITLE:	Implementation of Fixed and Sequential Multilevel Acceptance Sampling: The **R** Package MFSAS
AUTHOR:	Yalin Chen, B.Sc (McMaster University, Canada)
SUPERVISOR:	Dr. Aaron Childs
NUMBER OF PAGES:	x, 115

Abstract

Manufacturers and consumers often use acceptance sampling to determine the acceptability of a lot from an outgoing production or incoming shipment base on a sample. Multilevel acceptance sampling for attributes is applied when the product has multiple levels of product quality or multiple types of (mutually exclusive) possible defects.
The aim of this project is to develop an **R** package **MFSAS** which provides the tools to create, evaluate, plot, and display multilevel acceptance sampling plans for attributes for both fixed and sequential sampling. The Dirichlet recursive functions are used to calculate cumulative probabilities for several common multivariate distributions which are needed in the package.

Key words: acceptance sampling; multilevel quality control; operating characteristic; multinomial distribution; multivariate hypergeometric distribution; negative multinomial distribution; negative multivariate hypergeometric distribution; Dirichlet recursive functions.

Acknowledgements

Foremost, I am deeply grateful to my supervisor Dr. Aaron Childs for taking lots of time out of his busy schedule every week to provide me guidance and help throughout the course of my project. His great efforts to explain things clearly and simply, his inspiration, immense patience, good teaching, and lots of good ideas are solid foundation for this work. Furthermore, I very much appreciate his financial support for this project.

Besides my supervisor, I would like to express my great gratitude to my thesis committee, professors Román Viveros-Aguilera and Abdel El-Shaarawi, for agreeing to the task, and taking the time to read my thesis. Their insightful comments, questions, and encouragement are really appreciated.

I would also like to thank professors Narayanaswamy Balakrishnan and Román Viveros-Aguilera teaching me Statistics, and giving me help and inspiration during my graduate study.

I express my appreciation to professors Shui·Feng, Peter Macdonald, Ernie R. Mead and Fred M. Hoppe for making my education a extremely valuable experience.

My special thanks go out to Lihua Wang, Chang Ye, Yanyuen Liu, Xiaoyi Lin, Defen Peng, William Volterman, Ick Huh, Debanjan Mitra and Yuqing Bai for studying statistics with me and providing the support for my thesis presentation.

Last but not the least, I would like to thank my family, especially my son, for their understanding, love and support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Acceptance sampling is used to decide whether a lot from an incoming shipment or outgoing production should be accepted or rejected by making an inference about the lot quality based on a sample. It is an important aspect of statistical quality control.

Acceptance sampling can be dated back to the formation of the Inspection Engineering Department of Western Electric's Bell Telephone Laboratories in 1924. The first control chart appeared in 1924. The terminology of acceptance sampling (consumer's risk, producer's risk, probability of acceptance, OC curves, etc.) as well as lot tolerance percent defective (LTPD) sampling tables came out between 1925 and 1926. Average outgoing quality limit (AOQL) sampling tables were also presented in 1928.

In the 1930s, applications of acceptance sampling were used within Western Electric and elsewhere. The American Society of Mechanical Engineers, the American Society for Testing and Materials (ASTM), American Institute of Electrical Engineers, American Statistical Association, and the American Mathematical Society formed a Joint Committee for the Development of Statistical Applications in Development and Manufacturing in 1930. After that, Pearson (1935) developed British Standards Institution Standard Number 600, *Application of Statistical Methods to Industrial Standardization and Quality Control*, and Jennett and Welch (1939) published their paper on variables plans in England. In a doctoral dissertation called "Allowable average in sampling inspection", Romig (1939) presented the variables sampling plans along the lines of the Dodge-Romig tables which had been in use in Western Electric for some time in the USA.

Acceptance sampling was popularized by Dodge and Romig and originally applied by the U.S. military to the testing of bullets during World War II. During this period, Dodge and Romig (1941) published "Single sampling and double sampling inspection tables". These tables provided plans based on fixed consumer risk (LTPD protection) and plans for rectification (AOQL protection) which guaranteed specified protection after 100% inspection of rejected lots. In 1942, the Ordnance sampling tables of *Standard Inspection Procedures* were developed by the Army's Office of the Chief of Ordnance and later they grew into the Army Service Forces (ASF) tables of 1944 (U.S. Department of the Army, 1944). Dodge (1943) published "A sampling plan for continuous production" which developed an acceptance sampling plan for rectification inspection on a continuous sequence of products to assure consumer protection based on the maximum average quality the consumer would receive (AOQL protection). Statistical Research Group in Columbia University made some outstanding contributions which consisted of advancements in variables and attributes sampling in addition to sequential analysis. In 1948, this group published "Sampling Inspection" which contained a manual on sampling inspection prepared for the U.S. Navy Office of Procurement and Material. Bowker and Goode (1952) published a book "Sampling Inspection by Variables" which was a milestone in the development of variables sampling plans. After World War II many procedures have been developed, and now these statistical procedures are widely employed in many fields such as food inspection, industry quality control, etc.

More details of the historical development of acceptance sampling can be found in Dodge (1969a-c; 1970a) and in a series of papers published by the American Statistical Association (ASA, 1950) under the title *Acceptance Sampling.*

Acceptance sampling is divided into two major classifications:

- Attributes sampling, in which an inspected item has two or more levels of product quality (or multiple types of possible defects) or the number of nonconformities in an item are counted;
- Variable sampling, in which a characteristic in the inspected item is measured on a predetermined continuous scale.

Attributes sampling is the most common form of acceptance sampling, and will be assumed for the rest of the sections.

An appropriate sampling plan can be used to make an inference about whether the lot contains an acceptable proportion of each possible type of product defect; however, there is always a possibility that an incorrect decision is made from the sample. The probability of an incorrect decision is either

- the *Producer's Risk (PR)*, which is the probability that a lot which has the *Producer's Quality Level (PQL)*, at which the lot should be accepted most of the time, is rejected by the plan,

or

- the *Consumer's Risk (CR)*, which is the probability that a lot which has the *Consumer's Quality Level (CQL)*, at which the lot should be rejected most of the time, is accepted by the plan.

An effective sampling plan should be the one which can reduce both risks. Thus it should have a specified high probability of accepting a lot that the producer considers to be of good quality, and has a specified low probability of accepting a lot that the consumer considers to be of poor quality.

Once a sampling plan has been determined, the probability of accepting lots can be calculated for various levels of lot quality to obtain the *Operating Characteristic (OC)* function. For an up-to-date and complete reference on all aspects of acceptance sampling the reader is referred to Schilling and Neubauer (2009).

Many studies focus on two level sampling plans and their properties. Only a few papers investigate multilevel sampling plans and mainly concentrate on three level plans; see, for example, Cassady and Nachlas (2003).

For computation purposes, Kiermeier (2008) developed the **R** package **AcceptanceSampling**, which can be used when the sample size is fixed and there are $k = 2$ levels of product quality.

1.2 Objective

The objective of our work is to develop an R package **MFSAS**, which provides functionality for creating and evaluating acceptance sampling plans for attributes when there are k (≥ 2) levels of product quality. Plans can be multilevel fixed, or multilevel sequential.

In Chapter 2 of this thesis, we first introduce multivariate acceptance sampling and the multivariate distributions used to calculate acceptance probabilities. Then we explain the Dirichlet functions applied in this package, which are used for the calculation of cumulative probabilities for the multivariate distributions. We then discuss the implementation of the **MFSAS** package in Chapter 3 and give examples of its use in Chapter 4. Finally, in Chapter 5, we briefly discuss the pros and cons of the package and possible directions for future work.

The package documentation, including all help files, is given in Appendix A, and all of the code can be found in Appendix B.

Chapter 2

Multilevel Acceptance Sampling

Whether a lot from a manufacturer should be accepted or rejected is determined by making an inference about the lot quality based on a sample. The product quality can be described by classifying the product using more than two discrete levels in many circumstances. For example, a food product could be classified as good, marginal, or bad, depending on the concentration of harmful microorganisms in the product. Products in MIL-STD-105E (1989) are classified as critical defective, major defective, minor defective, or nondefective. Other examples of three or more classifications can be found in Cassady and Nachlas (2003), Bray, Lyon, and Burr (1973), Newcombe and Allen (1988), Thatcher and Clarke (1978), and Shapiro and Zaheda (1990).

Multilevel acceptance sampling can be used for such a multilevel product quality measure. In this project, multilevel acceptance sampling plans are divided into two types, fixed and sequential. A fixed sampling plan has a fixed sample size, and allows the user to make an accept/reject decision after inspecting a pre-specified number of items. In contrast to fixed sampling plans, the sample size in sequential sampling plans is a random variable. Sequential plans sample and inspect the items in the lot one at a time, and they can allow a decision to be reached more quickly in many cases.

2.1 Fixed Sampling

Suppose a sample of size n is selected from the lot. Let k (≥ 2) denote the number of different levels of product quality, one of which consists of nondefective or good items, and let X_i be the number of defectives of type i in the sample, for $i = 1, 2, \ldots, k-1$. A fixed sampling plan requires rejection numbers r_i with the property that the lot will be rejected if $X_i \geq r_i$ for any i in $\{1, 2, \ldots, k-1\}$. Thus, the probability of acceptance p_a is calculated as

$$p_a = P(X_1 \leq r_1 - 1,\ X_2 \leq r_2 - 1, \cdots,\ X_{k-1} \leq r_{k-1} - 1)\,.$$

If the sample is drawn from a population *with* replacement, or if the population is large compared to the sample size, then the multinomial distribution is used to calculate the acceptance probability. The multivariate hypergeometric distribution is used when sampling from a finite population *without* replacement.

2.1.1 Multinomial Distribution

If a sample of size n is drawn from a population whose k classes have probabilities $p_1, \ldots, p_{k-1}, p_k$, let $X_1, \ldots, X_{k-1}, X_k$ denote the number of observations drawn from each of the k classes. Then the cumulative probability of the multinomial distribution is given by

$$P(X_1 \leq x_1, \ldots, X_{k-1} \leq x_{k-1}) = \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} P(X_1 = y_1, \ldots, X_{k-1} = y_{k-1})$$

$$= \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} \frac{n!}{y_1! \ldots y_{k-1}! \left(n - \sum_{i=1}^{k-1} y_i\right)!} p_1^{y_1} \ldots p_{k-1}^{y_{k-1}} * \left(1 - \sum_{i=1}^{k-1} p_i\right)^{\left(n - \sum\limits_{i=1}^{k-1} y_i\right)}, \qquad (2.1)$$

where the sum is over all values of y such that $\sum\limits_{i=1}^{k-1} y_i \leq n$.

2.1.2 Multivariate Hypergeometric Distribution

If a sample of size n is drawn from a population of size N which has M_i objects of type i (for $i = 1, 2, \ldots, k$), let X_i be the number of objects of type i in the sample (for $i = 1, 2, \ldots, k$). Then the cumulative probability of the multivariate hypergeometric distribution is given by,

$$P(X_1 \leq x_1, \ldots, X_{k-1} \leq x_{k-1}) = \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} P(X_1 = y_1, \ldots, X_{k-1} = y_{k-1})$$

$$= \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} \frac{\binom{M_1}{y_1} \cdots \binom{M_{k-1}}{y_{k-1}} \binom{N - \sum_{i=1}^{k-1} M_i}{n - \sum_{i=1}^{k-1} y_i}}{\binom{N}{n}}, \tag{2.2}$$

where the sum is over all values of y such that $\sum\limits_{i=1}^{k-1} y_i \leq n$, and $y_i \leq M_i$.

2.2 Sequential Sampling

For "`sequential`" sampling, we require a cell quota m for the good items and cell quotas r_i for the each of the defective items, for $i = 1, 2, \ldots, k-1$. Sampling continues until either the number of good items or any of the $k-1$ types of defectives reaches its respective quota. If the former occurs first, then the lot is accepted; otherwise it is rejected.

Let X_i be the number of the i^{th} type of defectives selected in a sequence of trials before the cell quota m of good items is reached. Then, the acceptance probability p_a can be calculated as

$$p_a = P(X_1 \le r_1 - 1,\ X_2 \le r_2 - 1, \cdots,\ X_{k-1} \le r_{k-1} - 1)\,,$$

In a sequence of trials, if the sample is drawn from a population *with* replacement, or if the population is large compared to the sample size, then the negative multinomial distribution is used to calculate the acceptance probability. The negative multivariate hypergeometric distribution is used when selecting the sample (sequentially) from a finite population *without* replacement.

2.2.1 Negative Multinomial Distribution

Suppose that the population has $k-1$ different types of failures, with corresponding probabilities $p_1, \dots, p_{k-1}$ in each trial. Let $X_1, \dots, X_{k-1}$ denote the number of failures of each type that are selected in a sequence of trials before a target number m of successes is reached. Then the cumulative probability of the negative multinomial distribution is:

$$P(X_1 \le x_1, \dots, X_{k-1} \le x_{k-1}) = \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} P(X_1 = y_1, \dots, X_{k-1} = y_{k-1})$$

$$= \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} \frac{\left(\sum_{i=1}^{k-1} y_i + m - 1\right)!}{(m-1)!y_1! \dots y_{k-1}!} p_1^{y_1} \dots p_{k-1}^{y_{k-1}} \left(1 - \sum_{i=1}^{k-1} p_i\right)^m . \qquad (2.3)$$

2.2.2 Negative Multivariate Hypergeometric Distribution

Suppose that the population of size N has $k-1$ different types of failures represented $M_1, \dots, M_{k-1}$ times, respectively. Let $X_1, \dots, X_{k-1}$ denote the number of failures of each type that are selected in a sequence of trials before a target number m of successes is reached.

Then the cumulative probability of the negative multivariate hypergeometric distribution is:

$$P(X_1 \leq x_1, \cdots, X_{k-1} \leq x_{k-1}) = \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} P(X_1 = y_1, \cdots, X_{k-1} = y_{k-1})$$

$$= \sum_{y_1=0}^{x_1} \cdots \sum_{y_{k-1}=0}^{x_{k-1}} \frac{\binom{M_1}{y_1} \cdots \binom{M_{k-1}}{y_{k-1}} \binom{N - \sum_{i=1}^{k-1} M_i}{m-1}}{\binom{N}{m-1+\sum_{i=1}^{k-1} y_i}} \cdot \frac{N - m + 1 - \sum_{i=1}^{k-1} M_i}{N - m + 1 - \sum_{i=1}^{k-1} y_i}. \tag{2.4}$$

2.2.3 Expected Waiting Time

Under "`sequential`" sampling, the number of units actually inspected becomes a random variable. The expected waiting time - WT (or average sample number - ASN) for such procedures can be determined. Suppose that the population has $k-1$ different types of defectives whose realizations are represented as $d_1, d_2, \ldots, d_{k-1}$ with corresponding cell quotas $r_1, r_2, \ldots, r_{k-1}$, and good items represented as g with cell quota m. Let $X_1, \ldots, X_{k-1}$ denote the number of defects of each type that are selected in a sequence of trials.

Let A be the event that sampling stops as a result of the cell quota m of good items being reached first. Let B be the event that sampling stops as a result of any one of the cell quotas r_i of defective items being reached first (for $i = 1, 2, \ldots k-1$).
Then the expected waiting time can be calculated as:

$$\begin{aligned} E(WT) &= \sum \alpha \; P(WT = \alpha) \\ &= \sum \alpha \; P(WT = \alpha, \; A) + \sum \alpha \; P(WT = \alpha, \; B) \end{aligned}$$

For the negative multinomial distribution, the population has $k-1$ different types of failures, with corresponding probabilities $p_1, \ldots, p_{k-1}$ in each trial. Therefore,

$$P(WT = \alpha,\ A) = P(WT = \alpha,\ g \text{ last})$$

$$= P\left(\overbrace{\underbrace{d_1 \cdots d_1}_{X_1 < r_1} \ \underbrace{d_2 \cdots d_2}_{X_2 < r_2} \ \ldots \ \underbrace{d_{k-1} \cdots d_{k-1}}_{X_{k-1} < r_{k-1}} \ \underbrace{g \cdots g}_{X_k = m-1}}^{\text{first } \alpha - 1 \text{ trials, in any order}} \quad g \right)$$

$$= \sum_{x_1=0}^{r_1-1} \cdots \sum_{x_{k-1}=0}^{r_{k-1}-1} \frac{(\alpha-1)!}{x_1! \cdots x_{k-1}!(m-1)!} p_1^{x_1} \cdots p_{k-1}^{x_{k-1}} p_k^m, \tag{2.5}$$

where the sum is over all values of $X_1, \ldots, X_{k-1}$ such that $X_1 + \cdots + X_{k-1} + (m-1) = \alpha - 1$, and

$$P(WT = \ \alpha,\ B) = \sum_{i=1}^{k-1} P(WT = \alpha,\ d_i \text{ last})$$

$$= \sum_{i=1}^{k-1} P\left(\overbrace{\underbrace{d_1, \cdots d_1}_{X_1 < r_1} \ \ldots \ \underbrace{d_{i-1} \cdots d_{i-1}}_{X_{i-1} < r_{i-1}} \ \underbrace{d_i \cdots d_i}_{X_i = r_i - 1} \ \underbrace{d_{i+1} \cdots d_{i+1}}_{X_{i+1} < r_{i+1}} \ \ldots \ \underbrace{d_{k-1} \cdots d_{k-1}}_{X_{k-1} < r_{k-1}} \ \underbrace{g \cdots g}_{X_k < m-1}}^{\text{first } \alpha - 1 \text{ trials, in any order}} \quad d_i \right)$$

$$= \sum_{i=1}^{k-1} \sum_{x_1=0}^{r_1-1} \cdots \sum_{x_{i-1}=0}^{r_{i-1}-1} \sum_{x_{i+1}=0}^{r_{i+1}-1} \cdots \sum_{x_{k-1}=0}^{r_{k-1}-1} \sum_{x_k=0}^{m-1} \frac{(\alpha-1)!}{x_1! \cdots x_{i-1}! x_{i+1}! \cdots x_k! (r_i - 1)!}$$

$$p_1^{x_1} \cdots p_{i-1}^{x_{i-1}} p_{i+1}^{x_{i+1}} \cdots p_k^{x_k} p_i^{r_i}, \tag{2.6}$$

where $p_k = 1 - \sum_{i=1}^{k-1} p_i$, and the sum is over all values of $X_1, \ldots, X_{i-1},\ X_{i+1}, \ldots, X_k$ such that $X_1 + \cdots + X_{i-1} + X_{i+1} + \cdots + X_k + (r_i - 1) = \alpha - 1$.

For the negative multivariate hypergeometric distribution, the population of size N has $k-1$ different types of failures represented $M_1, \ldots, M_{k-1}$ times. Therefore,

$$P(WT = \ \alpha,\ A) = \sum_{x_1=0}^{\min(M_1, r_1-1)} \cdots \sum_{x_{k-1}=0}^{\min(M_1, r_{k-1}-1)}$$

$$\frac{\binom{N-\sum_{i=1}^{k-1} M_i}{m-1}\binom{M_1}{x_1}\cdots\binom{M_{k-1}}{x_{k-1}}}{\binom{N}{\alpha-1}} \frac{N-m+1-\sum_{i=1}^{k-1} M_i}{N-\alpha+1}, \tag{2.7}$$

where $\alpha \leq N$, $\alpha \leq m + \sum_{i=1}^{k-1} \min(M_i,\ r_i - 1)$, and the sum is over all values of $X_1, \ldots, X_{k-1}$ such that $X_1 + \cdots + X_{k-1} + (m-1) = \alpha - 1$,

$$P(WT = \alpha,\ d_i \text{ last}) = \sum_{x_1=0}^{\min(M_1, r_1-1)} \cdots \sum_{x_{i-1}=0}^{\min(M_{i-1}, r_{i-1}-1)}$$

$$\sum_{x_{i+1}=0}^{\min(M_{i+1}, r_{i+1}-1)} \cdots \sum_{x_{k-1}=0}^{\min(M_{k-1}, r_{k-1}-1)} \sum_{x_k=0}^{\min\left(N-\sum_{i=1}^{k-1} M_i, m-1\right)}$$

$$\frac{\binom{M_1}{x_1}\cdots\binom{M_{i-1}}{x_{i-1}}\binom{M_i}{r_i-1}\binom{M_{i+1}}{x_{i+1}}\cdots\binom{M_{k-1}}{x_{k-1}}\binom{N-\sum_{i=1}^{k-1} M_i}{x_k-1}}{\binom{N}{\alpha-1}} \frac{M_i - r_i + 1}{N-\alpha+1}, \tag{2.8}$$

where $i = 1, 2, \ldots, k-1$, $\alpha \leq N$, $\alpha \leq r_i + \min(N - \sum_{j=1}^{k-1} M_j,\ m-1) + \sum_{j=1, j\neq i}^{k-1} \min(M_j,\ r_j - 1)$, and the sum is over all values of $X_1, \ldots, X_{i-1},\ X_{i+1}, \ldots, X_k$ such that $X_1 + \cdots + X_{i-1} + X_{i+1} + \cdots + X_k + (r_i - 1) = \alpha - 1$.

Then, the expected waiting time for the multinomial case can be expressed as follows:

$$E(WT) = \sum_{\alpha=m}^{m+\sum_{i=1}^{k-1}(r_i-1)} \alpha\, P(WT=\alpha,\ g \text{ last}) + \sum_{i=1}^{k-1} \sum_{\alpha=r_i}^{m+\sum_{j=1}^{k-1}(r_j-1)} \alpha\, P(WT=\alpha,\ d_i \text{ last}), \quad (2.9)$$

and the expected waiting time for the multivariate hypergeometric case is given by

$$E(WT) = \sum_{\alpha=m}^{m+\sum_{i=1}^{k-1}\min(M_i, r_i-1)} \alpha\, P(WT=\alpha,\ g \text{ last})$$

$$+\sum_{i=1}^{k-1} \sum_{\alpha=r_i}^{r_i+\min\left(N-\sum_{j=1}^{k-1} M_j, m-1\right)+\sum_{j=1, j\neq i}^{k-1}\min(M_j, r_j-1)} \alpha\, P(WT=\alpha,\ d_i \text{ last}). \quad (2.10)$$

2.3 The Dirichlet Functions

Sobel, Uppuluri, and Frankowski (1977) introduced and developed the Dirichlet **J** function to calculate cumulative probabilities in the multinomal setting. Afterwards, the Dirichlet **HJ** function (Sobel and Frankowski, 1994) was similarly developed for the multivariate hypergeometric setting, the Dirichlet **D** function (Sobel, Uppuluri, and Frankowski, 1985) for the negative multinomal, and the Dirichlet **HD** function (Sobel and Frankowski, 1995; Childs, 2010) for the negative multivariate hypergeometric. These functions use multiple recurrence relations for the exact and highly efficient calculation of the cumulative probabilities for the corresponding multivariate distribution, thus eliminating the need for direct multiple summation. They also provide explicit formules for the expected waiting time, thus eliminating the need to compute equations (2.9) and (2.10) directly. Furthermore, it takes less time to compute the Dirichlet recursive functions than using multiple summation to calculate the cumulative probabilities in the R language. Therefore, we have used the Dirichlet **J**, **HJ**,

D, and **HD** functions to calculate cumulative probabilities for the multivariate distributions instead of direct multiple summation in the package **MFSAS** (see Appendix B.2).

2.3.1 The Dirichlet J Function

For the multinomial model, let b *blue cells* correspond to b different types of items in the population, each with the same probability p and the *sink* correspond to the remaining types of items with the probability $1-bp$ in the population. Let $J_p^{(b,j)}(r,n)$ denoted the probability that in a sample of size n taken from a large population, or with replacement from a finite population, j specified blue cells have exactly r observations and the remaining $b-j$ blue cells have fewer than r observations. It can be illustrated as in Figure 2.1.

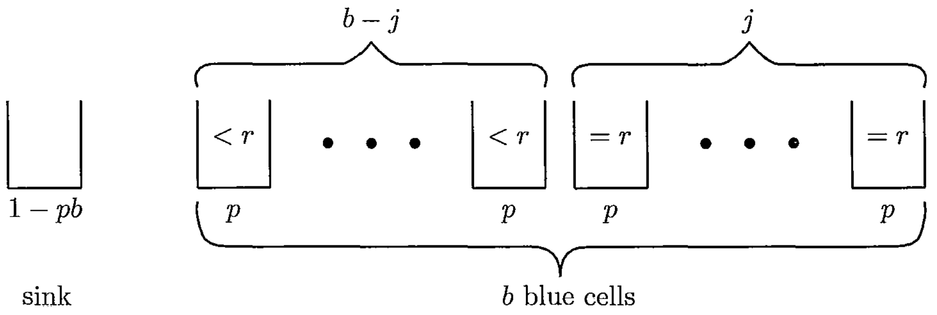

Figure 2.1: Blue cells with common probability p for the multinomial model

i.e., the probability $J_p^{(b,j)}(r,n)$ is defined as,

$$\begin{aligned} J_p^{(b,j)}(r,n) &= P(\text{exactly } r \text{ items in } j \text{ specified blue cells and less than } r \text{ items} \\ &\qquad \text{in the remaining } b-j \text{ blue cells in a sample of size } n). \end{aligned}$$

When $j=b$, we can evaluate the J function directly:

$$J_p^{(b,b)}(r,n) = P(\text{exactly } r \text{ items in } b \text{ specified cells in a sample of size } n)$$

$$= \frac{n!}{\underbrace{r!\ldots r!}_{b}(n-br)!}\underbrace{p^r\ldots p^r}_{b}(1-bp)^{n-br}$$

$$= \frac{n!}{(r!)^b(n-br)!}p^{br}(1-bp)^{n-br}, \quad \text{and is 0 for } n < br. \tag{2.11}$$

When $n = jr$, the probability can also be computed directly as

$$J_p^{(b,j)}(r, jr) = \frac{(jr)!}{(r!)^j}p^{jr}, \quad \text{and is 0 for } n < jr. \tag{2.12}$$

Using equations (2.11) and (2.12) as boundary conditions, the $J_p^{(b,j)}(r,n)$ can be calculated using a recurrence relation, which was developed by Sobel, Uppuluri, and Frankowski (1977), and is given in the following equation:

$$(n-jr)J_p^{(b,j)}(r,n) = n(1-jp)J_p^{(b,j)}(r,n-1) - (b-j)rJ_p^{(b,j+1)}(r,n). \tag{2.13}$$

In order to define the **J** function with vector arguments, suppose we have b blue cells with probability p_i (for $i = 1, 2, \ldots, b$) and a sink with probability $1 - \sum_{i=1}^{b} p_i$. Let $\vec{r} = (r_1, r_2, \ldots, r_b)$, and $\vec{p} = (p_1, p_2, \ldots, p_b)$ with $\sum_{i=1}^{b} p_i \le 1$. Let $J_{\vec{p}}^{(b)}(\vec{r};\, n)$ denote the probability that blue cell i will have less than r_i observations (for $i = 1, 2, \ldots, b$) in a sample of size n (see Figure 2.2).

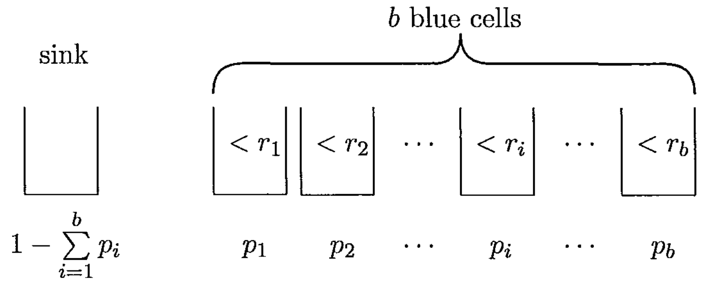

Figure 2.2: Blue cells with probability p_i having less than r_i observations for the multinomial model

Then $J_{\vec{p}}^{(b)}(\vec{r};\, n)$ can be calculated as follows:

$$\begin{aligned}
J_{\vec{p}}^{(b)}(\vec{r};\, n) &= P(< r_1 \text{ type } 1, < r_2 \text{ type } 2, \ \cdots , \ < r_b \text{ type b}) \\
&= \sum_{\alpha=0}^{r_1-1} P(= \alpha \text{ type } 1, < r_2 \text{ type } 2, \ \cdots , \ < r_b \text{ type b}) \\
&= \sum_{\alpha=0}^{r_1-1} P(< r_2 \text{ type } 2 \ , \ \cdots , \ < r_b \text{ type b} \mid = \alpha \text{ type } 1) P(= \alpha \text{ type } 1) \\
&= \sum_{\alpha=0}^{r_1-1} \binom{n}{\alpha} p_1^{\alpha} q_1^{n-\alpha} \ J_{\vec{p}_1/q_1}^{(b-1)}(\vec{r}_1;\, n-\alpha), \qquad (2.14)
\end{aligned}$$

where $\vec{p}_1 = \vec{p}$ and $\vec{r}_1 = \vec{r}$ with the first components removed and $q_1 = 1 - p_1$.

Equation 2.14 allows the dimension of $\vec{r}$ to be repeatedly reduced until one of the following boundary conditions can be used, $J_{\vec{p}}^{(1)}(\vec{r};n) = J_p^{(1,0)}(r,n)$ and $J_{\vec{p}}^{(b)}(\vec{r};\, n) = J_p^{(b,0)}(r,n)$ when $r_i = r$ and $p_i = p$ for $i = 1, 2, \ldots, b$.

The probability of acceptance p_a is then calculated as

$$p_a = P(X_1 \leq r_1 - 1, \ X_2 \leq r_2 - 1, \cdots , \ X_{k-1} \leq r_{k-1} - 1) \ = J_{\vec{p}}^{(k-1)}(\vec{r};\, n),$$

in the fixed sample multinomial setting, where $\vec{r} = (r_1, \ldots, r_{k-1})$ and $\vec{p} = (p_1, \ldots, p_{k-1})$.

2.3.2 The Dirichlet HJ Function

Let b blue cells correspond to b different types of items in the population, with the same number M of each item and let the sink cell correspond to the one remaining type of item which is represented $N - bM$ times, where N is the total number of items in population. Let $HJ_{M,\,N}^{(b,j)}(r,n)$ denoted the probability that in a sample of n observations, taken without replacement, j specified blue cells have exactly r observations and the remaining $b - j$ blue cells have fewer than r observations (see Figure 2.3).

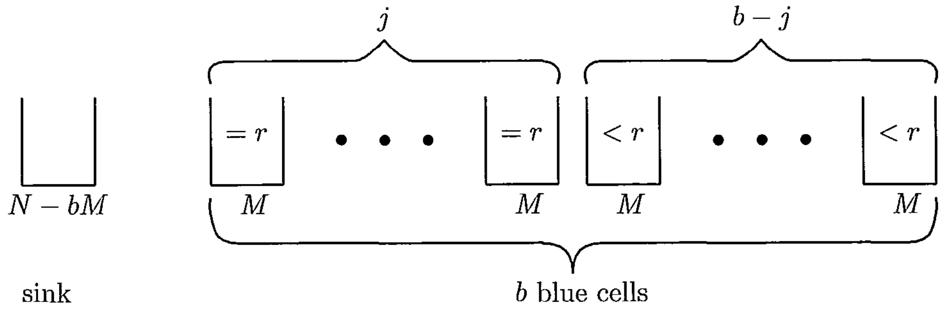

Figure 2.3: Blue cells with common M for the hypergeometric model

i.e., the probability $HJ_{M,N}^{(b,j)}(r,n)$ is defined as,

$$HJ_{M,N}^{(b,j)}(r,n) = P(\text{exactly } r \text{ items in } j \text{ specified blue cells and less than } r \text{ items in the remaining } b-j \text{ blue cells in a sample of size } n)$$

The recurrence relation for the hypergeometric model with commen M and r, was developed by Sobel and Frankowski (1994), and is given by

$$(n-jr)HJ_{M,N}^{(b,j)}(r,n) = n\left(1-\frac{j(M-r)}{N-n+1}\right)HJ_{M,N}^{(b,j)}(r,n-1)-(b-j)rHJ_{M,N}^{(b,j+1)}(r,n), \quad (2.15)$$

with the following boundary conditions,

$$HJ_{M,N}^{(b,b)}(r,n) = \frac{\binom{M}{r}^b\binom{N-bM}{n-br}}{\binom{N}{n}}, \quad \text{and is 0 for } n < br, \quad (2.16)$$

and

$$HJ_{M,N}^{(b,j)}(r,jr) = \frac{\binom{M}{r}^b}{\binom{N}{jr}}, \quad \text{and is 0 for } n < jr. \quad (2.17)$$

In order to define the **HJ** function with vector arguments, suppose we have b different types of items in the population represented $M_1, M_2, \ldots, M_b$, times, respectively, and a sink of size

$N - \sum_{i=1}^{b} M_i$. Let $\vec{r} = (r_1, r_2, \ldots, r_b)$, and $\vec{M} = (M_1, M_2, \ldots, M_b)$ with $\sum_{i=1}^{b} M_i \leq N$. Let $HJ^{(b)}_{\vec{M},N}(\vec{r};\, n)$ denote the probability that blue cell i will have less than r_i observations (for $i = 1, 2, \ldots, b$) in a sample of size n (see Figure 2.4).

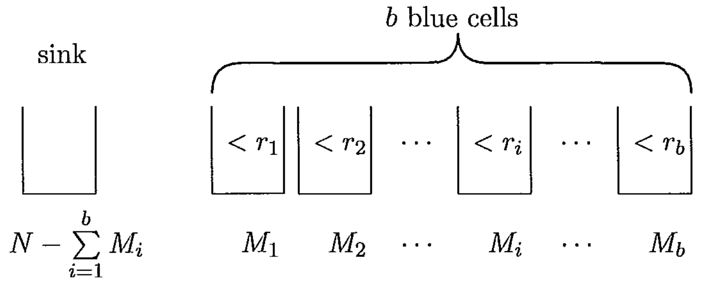

Figure 2.4: Blue cells with size M_i and cell quota r_i for the hypergeometric model

For the purpose of computing the vector form of the **HJ** function, we let $\vec{M}_1$ be the vector of cell sizes with the first cell size missing and $\vec{r}_1$ the $\vec{r}$ vector with the first component missing, i.e., $\vec{M}_1 = (M_2, \ldots, \ldots, M_b)$ and $\vec{r}_1 = (r_2, \ldots, r_b)$. Then the probability $HJ^{(b)}_{\vec{M},N}(\vec{r};\, n)$ is calculated as follows:

$$\begin{aligned} HJ^{(b)}_{\vec{M},N}(\vec{r};\, n) &= P(< r_1 \text{ type } 1, < r_2 \text{ type } 2, \ \cdots, \ < r_b \text{ type b}) \\ &= \sum_{\alpha=0}^{\min(M_1, r_1-1)} P(= \alpha \text{ type } 1, < r_2 \text{ type } 2, \ \cdots, \ < r_b \text{ type b}) \\ &= \sum_{\alpha=0}^{\min(M_1, r_1-1)} P(< r_2 \text{ type } 2\ , \ \cdots, \ < r_b \text{ type b} \mid = \alpha \text{ type } 1)\ P(= \alpha \text{ type } 1) \\ &= \sum_{\alpha=0}^{\min(M_1, r_1-1)} \frac{\binom{M_1}{\alpha}\binom{N-M_1}{n-\alpha}}{\binom{N}{n}} HJ^{(b-1)}_{\vec{M}_1, N-M_1}(\vec{r}_1;\, n-\alpha). \end{aligned} \tag{2.18}$$

Equation 2.18 allows the dimension of $\vec{r}$ to be repeatedly reduced until one of the following boundary conditions can be used, $HJ^{(1)}_{\vec{M},N}(\vec{r};\, n) = HJ^{(1,0)}_{M,N}(r, n)$, and $HJ^{(b)}_{\vec{M},N}(\vec{r};\, n) = HJ^{(b,0)}_{M,N}(r, n)$ for $M_i = M$ and $r_i = r$ for $i = 1, 2, \ldots, b$.

The probability of acceptance p_a is then calculated as

$$p_a = P(X_1 \leq r_1 - 1,\ X_2 \leq r_2 - 1, \cdots,\ X_{k-1} \leq r_{k-1} - 1) = HJ^{(k-1)}_{\vec{M},N}(\vec{r};\ n),$$

in the fixed sample size multivariate hypergeometric setting, where $r = (r_1, \ldots, r_{k-1})$ and $M = (M_1, \ldots, M_{k-1})$.

2.3.3 The Dirichlet D Function

In the negative multinomial setting, let b blue cells correspond to b different types of items in the population, each with the same probability p and let the *counting cell* correspond to the remaining type of item in the population, with the probability $1 - bp$. In this case, we select items one at a time until the counting cell reaches a pre-specified number m for the first time, which is called *stopping time.* Let $D_a^{(b,j)}(r,m)$ denoted the probability that j specified blue cells have exactly r observations and the remaining $b - j$ blue cells have fewer than r observations at stopping time, where $a = \frac{p}{p_0} = \frac{p}{1-bp}$. The setting is illustrated in Figure 2.5.

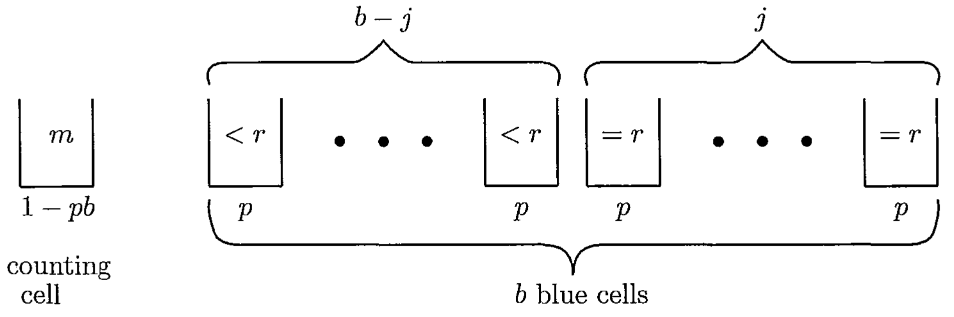

Figure 2.5: Blue cells with common probability p for the negative multinomial model

i.e., the probability $HJ^{(b,j)}_{M,N}(r,n)$ is defined as,

$$\begin{aligned} D_a^{(b,j)}(r,m) &= P(j \text{ specified blue cells have exactly } r \text{ observations and the remaining} \\ &\quad b - j \text{ blue cells have} < r \text{ observations at stopping time}). \end{aligned}$$

The recurrence relation for computing the **D** function was developed by Sobel, Uppuluri, and Frankowski (1985), and is as follows:

$$D_a^{(b,j)}(r,m) = \frac{1}{m+jr}[m(1+ja)D_a^{(b,j)}(r,m+1) + r(b-j)D_a^{(b,j+1)}(r,m)]. \qquad (2.19)$$

The first boundary condition is for $b = j$,

$$\begin{aligned} D_a^{(j,j)}(r,m) &= P\left(\overbrace{\underbrace{g\cdots g}_{m-1}\quad \underbrace{d_1\cdots d_1}_{r}\ \underbrace{d_2\cdots d_2}_{r}\ \ldots\ \underbrace{d_b\cdots d_b}_{r}}^{\text{in any order}}\quad g\right) \\ &= \frac{(m-1+br)!}{(r!)^b(m-1)!}p_0^m p^{br}, \end{aligned}$$

where g represents an item in counting cell and d_i, an item in the i^{th} blue cell (for $i = 1, 2, \ldots, b$).

Since $a = \frac{p}{p_0} = \frac{p}{1-bp}$, then $p_0 = \frac{1}{1+ab}$ and $p = \frac{a}{1+ab}$, therefore,

$$D_a^{(b,b)}(r,m) = \frac{\Gamma(m+br)}{(r!)^b\Gamma(m)}\left(\frac{a}{1+ba}\right)^{br}\left(\frac{1}{1+ba}\right)^m. \qquad (2.20)$$

The second boundary condition is for $m > r$, and is given by

$$D_a^{(b,j)}(r,m) = \frac{1}{\binom{m-1}{r}}\sum_{\alpha=1}^{r}\frac{\binom{m-\alpha-1}{r-\alpha}}{a^\alpha}D_a^{(b,j+1)}(r,m-\alpha). \qquad (2.21)$$

For the vector form of the **D** function, the setting is as follows. Suppose we have blue cells corresponding to b different types of items in population with probability p_i (for $i = 1, 2, \ldots, b$), and a counting cell of probability $1 - \sum_{i=1}^{b} p_i$. We take observations one at a time until the counting cell reaches a pre-specified number m of observations. Let $\vec{r} =$

$(r_1, r_2, \ldots, r_b)$, and $\vec{p} = (p_1, p_2, \ldots, p_b)$ with $\sum_{i=1}^{b} p_i \leq 1$ (for $i = 1, 2, \ldots, b$). Let $D^{(b)}_{\vec{p};p_0}(\vec{r}; m)$ denote the probability that the blue cell i will have less than r_i observations when the counting cell reaches m observations for the first time. The r_1's are referred to as *cell quotas* (See Figure 2.6).

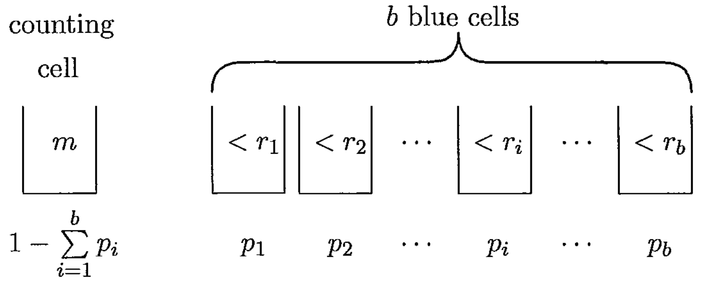

Figure 2.6: Blue cells with probability p_i and cell quota r_i for the negative multinomial model

Then the probability $D^{(b)}_{\vec{p};p_0}(\vec{r}; m)$ is calculated as follows,

$$\begin{aligned}
D^{(b)}_{\vec{p};p_0}(\vec{r}; m) &= P(< r_1 \text{ type } 1, < r_2 \text{ type } 2, \ \cdots, \ < r_b \text{ type } b \text{ at stopping time}) \\
&= \sum_{\alpha=0}^{r_1-1} P(= \alpha \text{ type } 1, < r_2 \text{ type } 2, \ \cdots, \ < r_b \text{ type } b \text{ at stopping time}) \\
&= \sum_{\alpha=0}^{r_1-1} P(< r_2 \text{ type } 2 \ , \ \cdots, \ < r_b \text{ type b} \mid = \alpha \text{ type } 1) P(= \alpha \text{ type } 1) \\
&= \sum_{\alpha=0}^{r_1-1} \frac{(m+\alpha-1)!}{\alpha!(m-1)!} \left(\frac{p_1}{p_1+p_0}\right)^{\alpha} \left(\frac{p_0}{p_1+p_0}\right)^{m} D^{(b-1)}_{\vec{p}_1;p_0+p_1}(\vec{r}_1; m+\alpha),
\end{aligned} \tag{2.22}$$

where $\vec{p}_1 = \vec{p}$ and $\vec{r}_1 = \vec{r}$ with the first components removed.

Equation 2.22 allows the dimension of $\vec{r}$ to be repeatedly reduced until one of the following boundary conditions can be used, $D^{(1)}_{\vec{p};p_0}(\vec{r}; m) = D^{(1,0)}_{\frac{p}{p_0}}(r, m)$, and $D^{(b)}_{\vec{p};p_0}(\vec{r}; m) = D^{(b,0)}_{\frac{p}{1-bp}}(r, m)$ when $r_i = r$ and $p_i = p$ for $i = 1, 2, \ldots, b$.

The probability of acceptance p_a is then calculated as

$$p_a = P(X_1 \le r_1 - 1,\ X_2 \le r_2 - 1, \cdots,\ X_{k-1} \le r_{k-1} - 1) \ = D_{\vec{p}}^{(k-1)}(\vec{r};\, m),$$

in the negative multinomial setting, where $\vec{r} = (r_1, \ldots, r_{k-1})$ and $\vec{p} = (p_1, \ldots, p_{k-1})$.

2.3.4 The Dirichlet HD Function

For the negative multivariate hypergeometric model, let b blue cells correspond to b different types of items in the population with the same number M of each, and the counting cell correspond to the one remaining type of item which is represented of $N - bM$ times, where N is the total number of items in population. Observations are taken one at a time without replacement until the counting cell reaches a pre-specified number m of the observations. Let $HD_{M,N}^{(b,j)}(r, m)$ denote the probability that j specified blue cells have exactly r observations and the remaining $b - j$ blue cells have fewer than r observations at stopping time.
(See Figure 2.7).

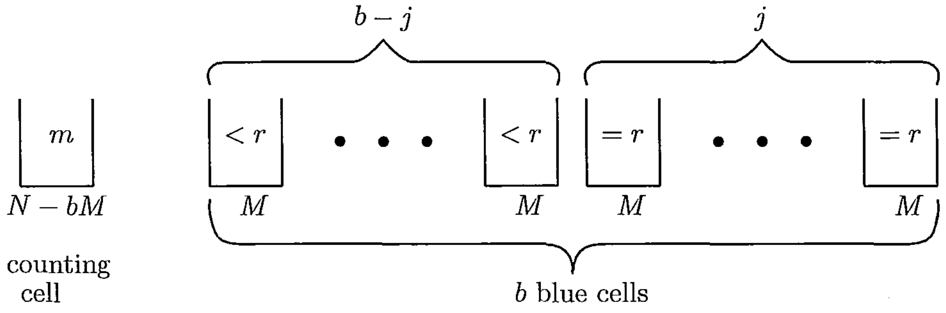

Figure 2.7: Blue cells with common M for the negative hypergeometric model

i.e., the probability $HD_{M,N}^{(b,j)}(r, m)$ is defined as

$$HD_{M,N}^{(b,j)}(r,m) = P(\text{exactly } r \text{ items in } j \text{ specified cells and less than } r \text{ items in the remaining } b-j \text{ blue cells at stopping time}).$$

The recurrence relation for the **HD** function with common M and r was developed by Sobel and Frankowski (1994), and is given by

$$HD_{M,N}^{(b,j)}(r,m) = \frac{1}{m+jr} \left\{ m\left[1+\frac{j(M-r)}{N-bM-m}\right] HD_{M,N}^{(b,j)}(r,m+1) + r(b-j)HD_{M,N}^{(b,j+1)}(r,m) \right\}. \quad (2.23)$$

The first boundary condition is for $b=j$, and is similar to the Dirichlet **D** function,

$$HD_{M,N}^{(b,b)}(r,m) = P\left(\overbrace{\underbrace{g\cdots g}_{m-1} \quad \underbrace{d_1\cdots d_1}_{r} \quad \underbrace{d_2\cdots d_2}_{r} \quad \cdots \quad \underbrace{d_b\cdots d_b}_{r}}^{\text{in any order}} \quad g \right)$$

$$= \frac{\binom{M}{r}^b \binom{N-bM}{m-1}}{\binom{N}{m-1+br}} \frac{N-bM-m+1}{N-br-m+1}$$

$$= \frac{m}{m+br} \frac{\binom{M}{r}^b \binom{N-bM}{m}}{\binom{N}{m+br}}, \quad (2.24)$$

where g represents an item in the counting cell and d_i is an item in the i^{th} blue cell (for $i = 1, 2, \ldots, b$).

The second boundary condition is for $m > r$, and is given by

$$HD_{M,N}^{(b,j)}(r,m) = \sum_{\alpha=0}^{r-1} \frac{m}{m+\alpha-r} \frac{\binom{M}{\alpha}\binom{N-bM}{m}}{\binom{M}{r}\binom{N-bM}{m+\alpha-r}} HD_{M,N}^{(b,j+1)}(r, m+\alpha-r). \quad (2.25)$$

For the vector form of the **HD** function, the setting is as follows. Suppose we have b different types of items in the population represented $M_1, M_2, \ldots, M_b$ times, respectively, and counting cell of size $N - \sum_{i=1}^{b} M_i$, where N is total items of the population. We take observations one at a time, without replacement, until the counting cell reaches a pre-specified number m of the observations. Let $\vec{M} = (M_1, \ldots, M_b)$ and $\vec{r} = (r_1, \ldots, r_b)$. Then $HD^{(b)}_{\vec{M};N}(\vec{r};\, m)$ is defined to be the probability that the blue cell i has less than r_i observations (for $i = 1, 2, \ldots, b$) at stopping time. The setting is illustrated as Figure 2.8.

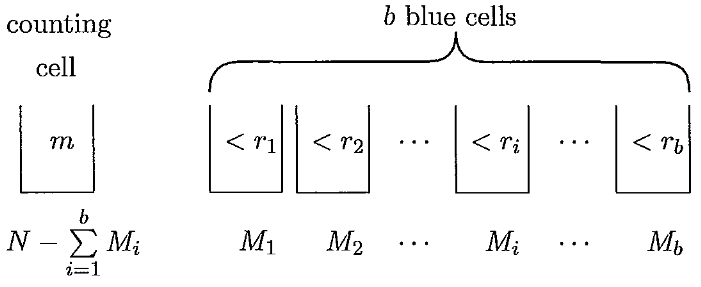

Figure 2.8: Blue cells with size M_i and cell quota r_i for the negative hypergeometric model

Let $\vec{M_1}$ be the vector of cell sizes with the first cell size missing and let $\vec{r}_1$ be the vector of cell quotas with the first cell quota missing, i.e., $\vec{M_1} = (M_2, \ldots, M_b)$ and $\vec{r}_1 = (r_2, \ldots, r_b)$. Then the probability $HD^{(b)}_{\vec{M};N}(\vec{r};\, m)$ is calculated as follows:

$$\begin{aligned}
HD^{(b)}_{\vec{M};N}(\vec{r};\, m) &= P(< r_1 \text{ type } 1, < r_2 \text{ type } 2,\ \cdots,\ < r_b \text{ type } b \text{ at stopping time}) \\
&= \sum_{\alpha=0}^{r_1-1} P(= \alpha \text{ type } 1, < r_2 \text{ type } 2,\ \cdots,\ < r_b \text{ type } b \text{ at stopping time}) \\
&= \sum_{\alpha=0}^{r_1-1} P(< r_2 \text{ type } 2\ ,\ \cdots,\ < r_b \text{ type b} \mid = \alpha \text{ type } 1)\ P(= \alpha \text{ type } 1) \\
&= \sum_{\alpha=0}^{r_1-1} \frac{\binom{M_1}{\alpha}\binom{N-\sum_{j=1}^{b} M_j}{m-1}}{\binom{N-\sum_{j=2}^{b} M_j}{m+\alpha-1}} \frac{N - \sum_{j=1}^{b} M_j - m + 1}{N - \sum_{j=2}^{b} M_j - \alpha + 1} HD^{(b-1)}_{\vec{M_1};N}(\vec{r}_1;\, m+\alpha)
\end{aligned}$$

$$= m\binom{N - \sum_{j=1}^{b} M_j}{m} \sum_{\alpha=0}^{r_1-1} \frac{\binom{M_1}{\alpha}}{\binom{N-\sum_{j=2}^{b} M_j}{m+\alpha}(m+\alpha)} HD^{(b-1)}_{\vec{M}_1;N}(\vec{r}_1;\, m+\alpha). \quad (2.26)$$

Equation 2.26 allows the dimension of $\vec{r}$ to be repeatedly reduced until one of the following boundary conditions can be used, $HD^{(1)}_{\vec{M},N}(\vec{r};\, m) = HD^{(1,0)}_{M,N}(r,m)$, and $HD^{(b)}_{\vec{M},N}(\vec{r};\, m) = HD^{(b,0)}_{M,N}(r,m)$ for $M_i = M$ and $r_i = r$ for $i = 1, 2, \ldots, b$.

The probability of acceptance p_a is then calculated as

$$p_a = P(X_1 \le r_1 - 1,\ X_2 \le r_2 - 1, \cdots,\ X_{k-1} \le r_{k-1} - 1) = HD^{(k-1)}_{\vec{M},N}(\vec{r};\, m),$$

where $\vec{M} = (M_1, \ldots, M_{k-1})$ and $\vec{r} = (r_1, \ldots, r_{k-1})$ in the negative multivariate hypergeometric setting.

2.3.5 Expected Waiting Time Using Dirichlet Recursive Function

Suppose the notations for calculating the expected waiting time are the same as section 2.2.3, and suppose that $b = k - 1$. Then

$$E(WT) = \sum \alpha\ P(WT = \alpha),$$

$$P(WT = \alpha) = P(WT = \alpha,\ g \text{ last}) + \sum_{i=1}^{b} P(WT = \alpha,\ d_i \text{ last}).$$

In the negative multinomial setting we can calculating the average sampling number using the Dirichlet **J** function as follows;

$$P(WT = \alpha,\ g \text{ last}) = \binom{\alpha - 1}{m - 1} p_0^m (1 - p_0)^{\alpha - m}\ J^{(b)}_{\frac{\vec{p}}{1-p_0}}(\vec{r};\, \alpha - m),$$

$$P(WT=\alpha,\ d_i \text{ last}) = \binom{\alpha-1}{r_i-1} p_i^m (1-p_i)^{\alpha-m}\ J^{(b)}_{\frac{(p_0,\vec{p}_i)}{1-p_i}}((m,\ \vec{r}_i);\ \alpha-r_i),$$

where $\vec{p} = (p_1, p_2, \ldots, p_b)$, $\vec{p}_i$ is the vector $\vec{p}$ with the i^{th} probability missing (for $i = 1, 2, \ldots, b$), and corresponding $\vec{r}_i$ the the vector of cell quotas with the i^{th} cell quota missing, i.e., $\vec{p}_i = (p_1, \ldots, p_{i-1}, p_{i+1} \ldots, p_b)$ and $\vec{r}_i = (r_1, \ldots, r_{i-1}, r_{i+1} \ldots, r_b)$.

Then the expected waiting time is given by

$$\begin{aligned} E(WT) &= \sum \alpha\ P(WT=\alpha) \\ &= \sum_{\alpha=m}^{m+\sum_{j=1}^{b}(r_j-1)} \alpha \binom{\alpha-1}{m-1} p_0^m (1-p_0)^{\alpha-m}\ J^{(b)}_{\frac{\vec{p}}{1-p_0}}(\vec{r};\ \alpha-m) \\ &+ \sum_{i=1}^{b} \sum_{\alpha=r_i}^{m+\sum_{j=1}^{b}(r_j-1)} \alpha \binom{\alpha-1}{r_i-1} p_i^m (1-p_i)^{\alpha-m}\ J^{(b)}_{\frac{(p_0,\vec{p}_i)}{1-p_i}}((m,\ \vec{r}_i);\ \alpha-r_i). \end{aligned} \quad (2.27)$$

Let $\vec{p}^* = (p_0 = 1 - \sum_{j=1}^{b} p_j, p_1, \ldots, p_b)$ and $\vec{r}^* = (r_0 = m, r_1, \ldots, r_b)$, $\vec{p}_i^*$ is the vector $\vec{p}^*$ with the i^{th} probability missing (for $i = 0, 1, 2, \ldots, b$), and $\vec{r}_i^*$ the vector of cell quotas with the i^{th} cell quota missing. Hence the equation (2.27) can be expressed more simply as

$$E(WT) = \sum_{i=0}^{b} \sum_{\alpha=r_i}^{\sum_{j=0}^{b} r_j - b} \alpha \binom{\alpha-1}{r_i-1} p_i^m (1-p_i)^{\alpha-m}\ J^{(b)}_{\frac{\vec{p}_i^*}{1-p_i}}(\vec{r}_i^*;\ \alpha-r_i). \quad (2.28)$$

Sobel, Uppuluri, and Frankowski (1985) have shown that the above expression for the average sample number can be calculated using the Dirichlet **D** function as follows;

$$E(WT) = \frac{m}{p_0} D^{(b)}_{\frac{\vec{p}}{p_0}}(\vec{r};\ m+1) + \sum_{i=1}^{b} \frac{r_i}{p_i} D^{(b)}_{\frac{(p_0,\ \vec{p}_i)}{p_i}}((m,\ \vec{r}_i);\ r_i+1)$$

$$= \sum_{i=0}^{b} \frac{r_i}{p_i} D^{(b)}_{\frac{\vec{p}_i}{p_i}}(\vec{r}_i^*;\ r_i+1). \quad (2.29)$$

Equation (2.29) is used to calculate the average sample number for sequential sampling in the multinomial setting in the package **MFSAS** (see Appendix B.3).

In the negative multivariate hypergeometric setting we can similarly use the Dirichlet **HJ** and **HD** functions.

Let $\vec{M}_i$ be the vector of cell sizes with the i^{th} cell size missing and $\vec{r}_i$ the vector of cell quotas with the i^{th} cell quota missing, i.e., $\vec{M}_i = (M_1, \ldots, M_{i-1}, M_{i+1} \ldots, M_b)$ and $\vec{r}_i = (r_1, \ldots, r_{i-1}, r_{i+1} \ldots, r_b)$. Then the expected waiting time is given by

$$E(WT) = \sum \alpha \; P(WT = \alpha)$$

$$= \sum_{\alpha=m}^{\min\left(m+\sum_{j=1}^{b}(r_j-1),\, N,\, m+\sum_{j=1}^{b} M_j\right)} \alpha \frac{\binom{N-\sum_{j=1}^{b} M_j}{m-1}\binom{\sum_{j=1}^{b} M_j}{\alpha-m}}{\binom{N}{\alpha-1}}$$

$$\times \frac{N-\sum_{j=1}^{b} M_j - m + 1}{N-\alpha+1} HJ^{(b)}_{\vec{M},\sum_{j=1}^{b} M_j}(\vec{r}; \alpha - m)$$

$$+\sum_{i=1}^{b} \sum_{\alpha=r_i}^{\min\left(m+\sum_{j=1}^{b}(r_j-1),\, N,\, N-M_i+r_i\right)} \alpha \frac{\binom{M_i}{r_i-1}\binom{N-M_i}{\alpha-r_i}}{\binom{N}{\alpha-1}}$$

$$\times \frac{M_i - r_i + 1}{N-\alpha+1} HJ^{(b)}_{\left(N-\sum_{j=1}^{b} M_j, \vec{M}_i\right); N-M_i}((m,\; \vec{r}_i); \alpha - r_i). \tag{2.30}$$

Let $\vec{M}^* = (M_0 = N - \sum_{j=1}^{b} M_j,\, M_1, \ldots, M_b)$ and $\vec{r}^* = (r_0 = m, r_1, \ldots, r_b)$, $\vec{M}_i^*$ is the vector $\vec{M}^*$ with the i^{th} cell size missing (for $i = 0, 1, 2, \ldots, b$), and $\vec{r}_i^*$. is the vector of cell quotas with the i^{th} cell quota missing. Hence the equation (2.30) can be expressed more simply as

$$E(WT) = \sum_{i=0}^{b} \sum_{\alpha=r_i}^{\min\left(\sum_{j=0}^{b}(r_j-1),\, N,\, N-M_i+r_i\right)}$$

$$\alpha \frac{\binom{M_i}{r_i-1}\binom{N-M_i}{\alpha-r_i}}{\binom{N}{\alpha-1}} \times \frac{M_i - r_i + 1}{N-\alpha+1} HJ^{(b)}_{\vec{M}_i^*;\, N-M_i}(\vec{r}_i^*; \alpha - r_i). \tag{2.31}$$

Childs (2010) showed that above expression for the average sample number can be calculated using hte Dirichlet **HD** function as follows:

$$\begin{aligned} E(WT) =& \frac{m(N+1)}{N+1-\sum_{i=1}^{b} M_i} HD^{(b)}_{\vec{M};\, N+1}(\vec{r};\, m+1) \\ &+ \sum_{i=1}^{b} \frac{r_i(N+1)}{M_i+1} HD^{(b)}_{(N-\sum_{j=1}^{b} M_j,\, \vec{M});\, N+1}((m,\, \vec{r});\, r_i+1) \\ =& \sum_{i=0}^{b} \frac{r_i(N+1)}{M_i+1} HD^{(b)}_{\vec{M}_i^*;\, N+1}(\vec{r}_i^*;\, r_i+1). \end{aligned} \tag{2.32}$$

Equation (2.32) is used to calculate the average sample number for sequential sampling in the multivariate hypergeometric setting in the package **MFSAS** (see Appendix B.3).

Chapter 3

Implementation - The R Package MFSAS

The MFSAS package is based on formal S4 classes and methods. It provides functionality for creating, evaluating, and plotting k-level acceptance sampling plans for attributes according to different distributional assumptions.

3.1 Object Classes

The package consists of a virtual class, `Ocmult`. In this class the two parameters are `type` (the type of distribution) and `stype` (the type of sampling). The distributions that can be specified for `type` are

- `multinomial` which is used if the number of items in the lot is assumed to be large relative to the sample size, or if the sample is taken with replacement.
- `hypergeom` which is used when the lot is finite and the sample is taken without replacement.

The types of sampling that can be specified with stype are

- fixed for fixed sampling, in which a sample of size n is selected from the lot. For this type of sampling, calculations are based on either the multinomial or multivariate hypergeometric distribution, depending on the value specified for type.
- sequential for sequential sampling, in which items are selected one at a time for inspection. Here calculations are based on either the negative multinomial or negative multivariate hypergeometric distribution, depending on the value specified for type.

The two actual classes, Ocmult.multinomial and Ocmult.hypergeom are derived from Ocmult (see Figure 3.1).

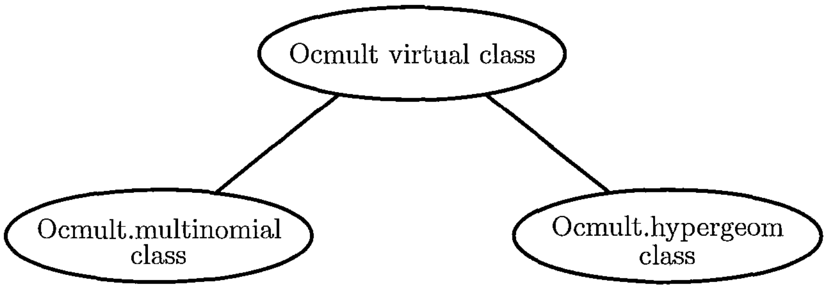

Figure 3.1: Class structure

Both classes contain the Ocmult virtual class, hence its slots. Objects of the two classes can be generated by the Ocmult function which takes the following arguments.

rn: A vector of length k-1 consisting of rejection numbers for fixed sampling, or cell quotas for the defective items for sequential sampling.

pd: A matrix with k-1 columns, whose rows contain the proportions of each type of defective in the population.

stype: The type of sampling.

`type`: The type of distribution on which the plans are based.

`...` : Additional arguments which depend on the distribution to be used and the type of sampling. When `type="hypergeom"` the lot size `N` needs to be specified. Since `pd*N` is a matrix containing the actual number of each type of defective in the lot, its entries must be nonnegative integers. The `stype="fixed"` needs `n` (the sample size), and for `stype="sequential"` `m` (the cell quota for good items) must be provided.

The new object is created and returned after the arguments are initialized and validated by the initialize and validation functions which are part of the class building.

The R code to create all of the above mentioned classes, as well as the initialization, validation, plot and summary methods discussed in the following sections can be found in Appendix B.1.

3.2 Initialize and Validation Methods

When creating an object, the initialize function creates the value for each argument according to the assumed distribution. Then the validation functions for the virtual class and actual classes validate if the sampling plan makes sense for the specific distribution. The validation functions for the virtual class applies to both actual classes because of inheritance.

The checks in the validation function for the virtual class `Ocmult` are:

- The vector `rn` contains no NA's;
- The values in the vector `rn` are greater than zero;
- The matrix pd contains no NA's;
- None of the entries in the matrix `pd` are less than 0;

- The sum of the values in each row of the matrix `pd` is not greater than 1;
- The length of the vector `rn` is equal to the number of columns in the matrix `pd`.

In the validation functions for the actual class `Ocmult.multinomial`, the arguments are validated as follows:

- For fixed sampling, the checks are
  - The sample size `n` contains no NA;
  - The value of `n` is greater than 1;
  - None of the values in `rn` are greater than `n`.
- For sequential sampling, the additional checks are
  - The cell quota `m` for good items contains no NA;
  - The cell quota `m` for good items is greater than 0.

In addition to the above, the validation function for the actual class `Ocmult.hypergeom` checks the following:

- The population size `N` contains no NA;
- The value of `N` is greater than 0;
- The length of `N` is equal to 1;
- The entries in the matrix of `pd*N` are all integers;
- The length of the vector `rn` is less then `N`.

3.3 Plot Methods

The operating characteristic (OC) function behavior of 2-level acceptance sampling plans can be presented by plotting the 2-dimensional OC curve corresponding to the sampling plans. In this package plot methods have been created for the actual classes with the proportion of defectives `pd` on the horizontal axis and the probabilities of acceptance `pa` on the vertical axis in the graph. The signatures in the plot methods are

- `signature(x = "Ocmult.multinomial", y = "missing");`
- `signature(x = "Ocmult.hypergeom", y = "missing");`
- `signature(x="numeric", y="Ocmult.multinomial");`
- `signature(x="numeric", y="Ocmult.hypergeom").`

For the latter two signatures, the plot is of the probabilities of acceptance `pa` against a numerical variable, which can be supplied by the user, instead of the proportion of defectives `pd`.

To plot the OC curve corresponding to the sampling plan, only an object of the particular class needs to be specified and all relevant details are extracted from the object (see Chapter 4 for examples).

The OC function behavior of 3-level acceptance sampling plans can be presented by two types of plots, *prosp* and *contour*. In the `prosp` methods for the actual classes, the proportions of defectives `pd` are on the two horizontal axes `x`, `y` and the probabilities of acceptance `pa` are on the vertical axis `z` in the graph. The signatures in the prosp methods are

- `signature(x = "Ocmult.multinomial");`

- signature(x = "Ocmult.hypergeom").

In the contour methods for the actual classes, the signatures are the same as the prosp method. In the contour graph, the curves represent the probabilities of acceptance pa for specific values corresponding the proportions of defectives pd on the two axes x, y. An example is provided in Chapter 4.

3.4 Summary Methods

The generic summary method is used to summarize the object. The show method gives a brief summary of the supplied object. For fixed sampling it displays the type of distribution, the sample size n, and the rejection number(s) rn. For sequential sampling, it displays the type of distribution and the cell quotas m and rn for the good and the defective item(s). If the type of distribution is hypergeometric, then the population size N is also shown.

The summary method shows the same detail as the show method by default, but provides the additional logical argument detail. If detail=TRUE, then all the information for the object is printed, including all values of pd and the corresponding values of pa as well as ASN (average sampling number) for sequential sampling. Examples of these methods are given in Chapter 4.

3.5 Assessing a Sampling Plan

The function assess.multi can be used to assess whether a sampling plan can meet specific criteria, and is given in Appendix B.4. The two types of criteria, the *Producer's Risk Point (PRP)* and the *Consumer's Risk Point (CRP)*, can be specified singly or together by the arguments PRP and CRP, respectively. The parameters of the plan are consistent with the classes (see Section 3.1).

Both risk points are vectors of length k which contain two parts. The first part consists of k-1 elements which represent the quality level (equivalent to a row of pd). The second part is the kth element which represents the corresponding probability of acceptance (equivalent to pa).

For the Producer's Risk Point to be met, the probability of acceptance of the plan must be *at least* equal to the value specified by the user in PRP[k] corresponding to the PQL, the producer's quality level (PRP[-k]). Note that the producers's risk is at most 1− PRP[k].

For the Consumer's Risk Point to be met, the probability of acceptance of the plan must be *at most* equal to the value specified by the user in CRP[k] corresponding to the CQL, the consumer's quality level (CRP[-k]).

The argument print in the function assess.multi indicates whether a summary of the assessment should be printed. The default value is print=TRUE.

3.6 Finding a Sampling Plan

The function find.multi.plan allows the user to find a multilevel acceptance sampling plan which meets specified producer and consumer risk points (see Section 3.5), and can be found in Appendix B.5. Both points must be specified in the function and the CRP must have worse quality than the PRP.

In the case of type="multinomial", only the PRP and CRP need to be specified. For type ="hypergeom", the additional argument N (the lot size) must be provided.

For fixed sampling, the function finds the smallest sample size n and the corresponding rejection numbers for each type of defective which will meet the PRP and CRP requirements.

The process of find plan is through trial starting with $n = 1$ and increasing n until the appropriate plan is found. The rejection numbers for the defectives change appropriately at each step. For the `sequential` sampling, the function finds the smallest quota m for the good items and the corresponding cell quotas for each type of defective, which will meet the PRP and CRP requirements. The process is through trial starting with $m = 1$ and increasing m until the appropriate plan is found. The cell quotas for defectives change appropriately at each step but do not exceed the quota for the good items.

3.7 Cumulative Distribution Functions

The **MFSAS** package also includes functions to calculate the cumulative distribution functions (`CDFs`) for the multinomial, negative multinomial, multivariate hypergeometric, and negative multivariate hypergeometric distributions which are required for the calculation of acceptance probabilities for the multilevel sampling plans in this package. Chapter 2 contains a brief description, and the reader is referred to Johnson, Kotz, and Balakrishnan (1997) for further details about these distributions. The R code for all of the CDF's can be found in Appendix B.3, and is briefly described below.

- `pmultinom` - The multinomial CDF

  `pmultinom(x, size = n, prob = p)` is a function for the calculation of cumulative probabilities [equation (2.1)] for the multinomial distribution when a sample of size n is drawn from a population whose k classes have probabilities $p_1, \ldots, p_{k-1}, p_k$.
  Here `x` is a vector of length $k-1$ containing the number of observations drawn from $k-1$ of the classes and p is a vector of length $k-1$ specifying the probability for $k-1$ of the classes.
  `pmultinom` is computed using recursive algorithms for the Dirichlet **J** function in this

package. Section 2.3.1 gives a short explanation; for further details, see Sobel, Uppuluri, and Frankowski (1977) and Sobel and Frankowski (2004).

- `pnmultinom` - The negative multinomial CDF

  `pnmultinom(x, m, prob = p)` is a function for calculating cumulative probabilities for the negative multinomial distribution [equation (2.3)]. The vector `x` of length $k-1$ contains the number of failures of each type, with corresponding probabilities in each trial given in the `p` vector, that are selected in a sequence of trials before a target number `m` of successes is reached.

  `pnmultinom` is computed using recursive algorithms for the Dirichlet **D** function described in section 2.3.3, and developed by Sobel, Uppuluri, and Frankowski (1985) and Sobel and Frankowski (2004).

- `pmultihyper` - The multivariate hypergeometric CDF

  `pmultihyper(x, n, M, N)` is a function to calculate cumulative probabilities for the multivariate hypergeometric distribution [equation (2.2)] when a sample of size `n` is drawn from a population of size `N`, which has M_i objects of type i (for $i = 1, 2, \ldots, k$), without replacement. Here `x` is a vector of length $k - 1$ containing the number of objects of type i in the sample for $i = 1, 2, \ldots, k-1$, and `M` is a vector of length $k - 1$ containing the total number of objects in each of $k - 1$ of the classes.

  `pmultihyper` is computed using recursive algorithms for the Dirichlet **HJ** function described in section 2.3.2 and developed by Sobel and Frankowski (1994).

- `pnmultihyper` - The negative multivariate hypergeometric CDF

  `pnmultihyper(x, m, M, N)` is a function for calculating cumulative probabilities for the negative multivariate hypergeometric distribution [equation 2.4]. The vector `x` of length of $k-1$ contains the number of failures of each type that are selected in a sequence of trials without replacement from a population of size `N`, which has M_i failures of type

i (for $i = 1, 2, \ldots, k-1$), before a target number `m` of successes is reached.

`pnmultihyper` is computed using recursive algorithms for the Dirichlet **HD** function described in section 2.3.4, and developed by Childs (2010) and Sobel and Frankowski (1995).

Chapter 4

Examples

The package can be loaded as

```
   > library(MFSAS)
```

4.1 Creating `Ocmult` Plans

After the package is loaded, a new object of the sampling plan can be created by using the `Ocmult` function. For example, a sampling plan with $n = 30$ and $rn = (2, 4, 3)$ for a large lot size can be obtained as follows:

```
> p.mn <- Ocmult(rn=c(2,4,3), n=30)
> p.mn
  4-Level Acceptance Sampling Plan Multinomial:
 Sample size:    30
 Rej. Number(s): 2 4 3
```

For a finite population of size $N = 100$, type="h" should be specified. If the sample size is $n = 15$ and $rn = (2, 3)$, then, the sampling plan is obtained as:

```
> p.mh<- Ocmult(rn=c(2,3), n=15, N=100, type="h")
> p.mh
  3-Level Acceptance Sampling Plan Multivariate Hypergeom: N = 100
 Sample size:     15
 Rej. Number(s): 2 3
```

When stype="s" is given, the target number m of good items must be specified. If $k = 2$, and we continue to sample until we obtain either 5 good items or 3 defectives then the sequential sampling plan can be created as follows:

```
> p.nmn <- Ocmult(rn=3, m=5, stype="s", pd=seq(0, 1, 0.01))
> p.nmn
  2-Level Sequential Acceptance Sampling Plan Negative Multinomial:
 Acc. Number:     5
 Rej. Number(s): 3
```

4.2 Plotting Sampling Plans

The OC curve or surface can be plotted when the sampling plans are 2-level or 3-level.

For the 2-level sequential sampling plan with type="multinomial" given in the example above, the plot is shown in Figure 4.1:

```
> plot(p.nmn)
```

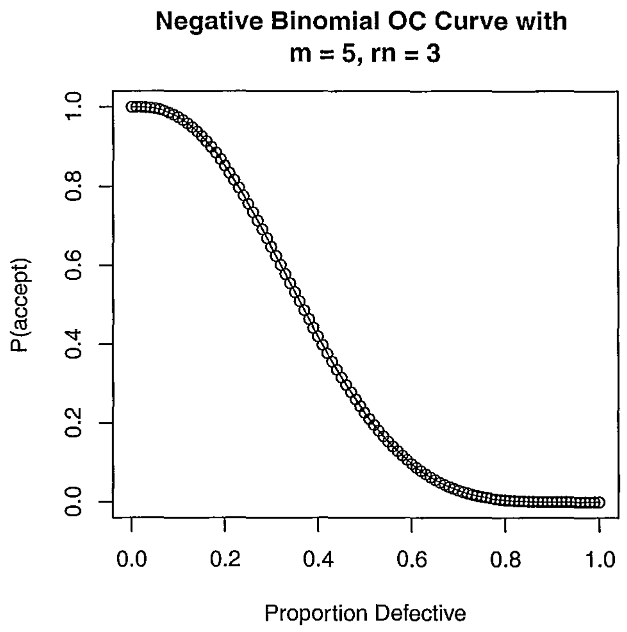

Figure 4.1: OC curve for the negative binomial distribution

All arguments for the standard plot method can be passed directly to the generic `plot` method in this package. `type="o"` in the standard plot type is the default value to show the OC curve using both lines and points for the binomial and negative binomial distributions. `type="p"` is the default value used in order to show the OC curve using only points for the hypergeometric and negative hypergeometric distributions.

The following example produces the graphs given in Figure 4.2.

```
x.mn <- Ocmult(n=8, rn=3, pd=seq(0, 0.22, 0.01))
x.mh <- Ocmult(n=8, rn=3, N=50, type="h", pd=seq(0, 0.2, 0.02))
x.nmn <- Ocmult(rn=2,m=4, stype="s", pd=seq(0, 0.22, 0.01))
x.nmh <- Ocmult(rn=2, m=4, N=50, type="h", stype="s", pd=seq(0, 0.2, 0.02))
main = "Fixed Sampling Plan \nn = 8, rn = 3"
plot(x.mn, type="l", xlim=c(0, 0.2), ylim=c(0.75, 1), main=main)
```

```
grid(lty="solid")
points(x.mh@pd, x.mh@pa, col = 3)
legend(0.01, 0.81, c("binomial", "hypergeometric"), col = c(1,3),
   lty = c(1, -1), pch = c(-1, 1), bg = 'gray95')
main = "Sequential Sampling Plan \nm = 4, rn = 2"
plot(x.nmn, type="l", xlim=c(0, 0.2), ylim=c(0.7, 1), main=main)
grid(lty="solid")
points(x.nmh@pd, x.nmh@pa, col= 3 )
legend(0.01, 0.77, c("negative binomial", "negative hypergeometric"),
   col = c(1,3), lty = c(1,-1),  pch = c(-1, 1), bg = 'gray96')
```

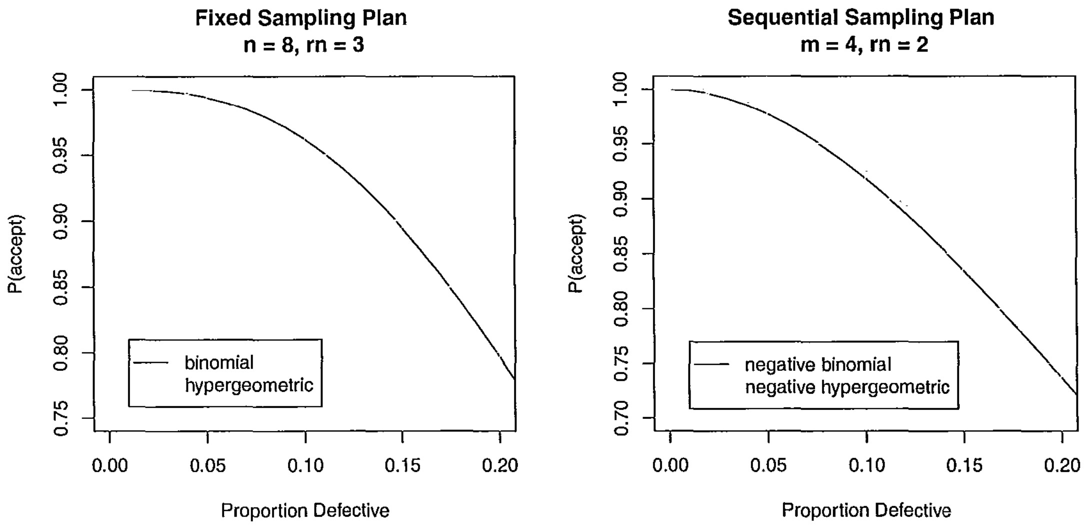

Figure 4.2: OC curve for fixed and sequential sampling

Using the 3-level sampling plan p.mh from Section 4.1, the plots for fixed sampling with type="hypergeom" are given in Figure 4.3, and are obtained as follows:

```
> persp(p.mh, cex.main = 1)
> contour(p.mh, cex.main = 1)
```

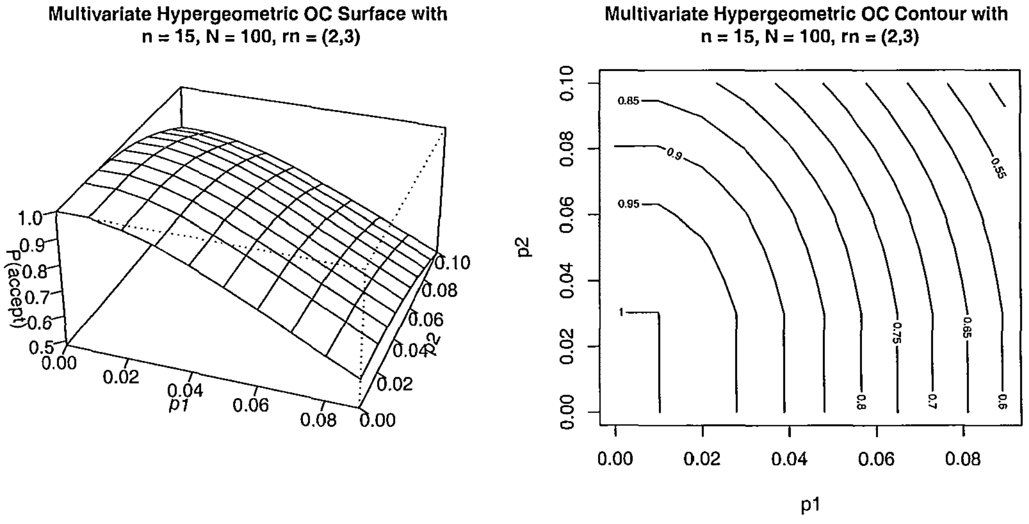

Figure 4.3: OC surface and contour curve for the multivariate hypergeometric distribution with $N = 100$, $n = 15$, $rn = (2, 3)$

All arguments for the persp and contour methods can be passed directly to the generic `persp` and `contour` methods in this package, respectively. For example in Figure 4.4 we use light blue color for the surface, and 4 contours levels for the contour plot,

```
> px <- as.matrix(expand.grid(seq(0,0.2, 0.01),seq(0, 0.8, 0.04)))
> p.multinom <- Ocmult(n=15,rn=c(2,10), pd=px)
> persp(p.multinom, col="light blue")
> contour(p.multinom, nlevel=4)
```

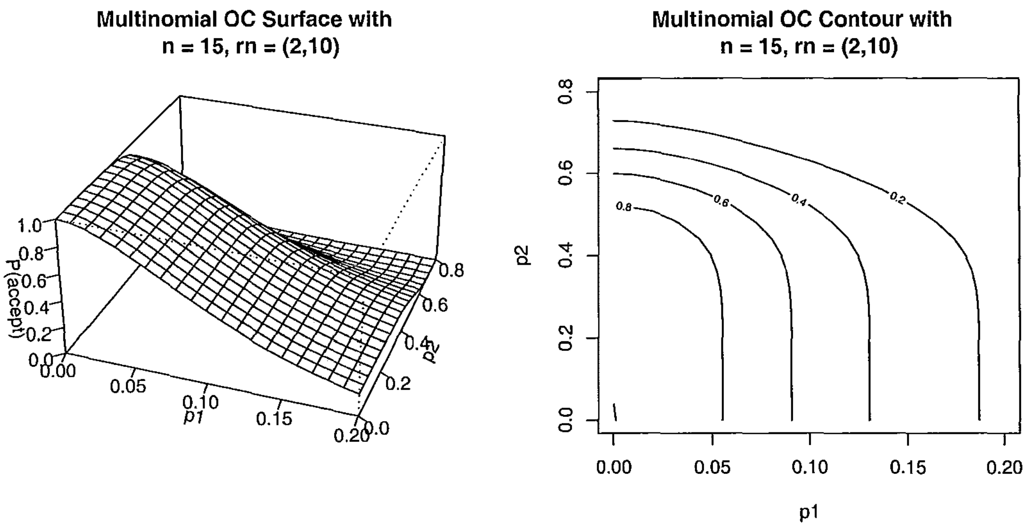

Figure 4.4: OC surface and contour curve for the multinomial distribution with $n = 15$, $rn = (2, 10)$

4.3 Sampling Plan Summary

The summary method gives a summary of the sampling plan with an option for detailed output, as in the following example.

```
> px <- matrix(c(seq(0, 0.5, 0.1), seq(0, 0.2, 0.04)), ncol=2)
> p.multinom <- Ocmult(n=30,rn=c(3,4), pd=px)
> summary(p.multinom ,detail=TRUE )
  3-Level Acceptance Sampling Plan Multinomial:
 Sample size:    30
 Rej. Number(s): 3 4


 Detailed acceptance probabilities:
   type 1  type 2  P.nondef  P(accept)
      0.0    0.00      1.00  1.0000000
```

```
0.1    0.04      0.86  0.3963991

0.2    0.08      0.72  0.0304085

0.3    0.12      0.58  0.0005591

0.4    0.16      0.44  0.0000017

0.5    0.20      0.30  0.0000000
```

Note that pd is a matrix with k−1 columns, where each row contains the proportions for each type of defective in the population. However, if a vector is provided, then it is converted to a matrix by row according to the length of rn. The vector will be truncated (with a warning) if its length is not an integer multiple of the length of rn. An example follows:

```
> px <-c(0.02,0.06,0.04,0.06,0.08,0.02,0.02,0.08,0.04,0.10)
> pmh <- OCmult(rn=c(2,3,2,4), n=20, N=100, pd=px, type="h")
Warning message:
In .local(.Object, ...) :
  The length of the pd vector should be an integer
multiple of the length of the rn vector.
The truncated pd in use is:
0.02  0.06  0.04  0.06  0.08  0.02  0.02  0.08
> summary(pmh ,detail=TRUE )
  5-Level Acceptance Sampling Plan Multivariate Hypergeom: N = 100
 Sample size:    20
 Rej. Number(s): 2 3 2 4

 Detailed acceptance probabilities:
   type 1  type 2  type 3  type 4  P.nondef  P(accept)
     0.02    0.06    0.04    0.06      0.82  0.7023403
     0.08     0.02     0.02     0.08        0.80  0.4485896
```

4.4 Assessing a Sampling Plan

The `assess.multi` function is used to assess a sampling plan given the PRP and/or CRP (see Section 3.5 for a description). For example, suppose we want a 3-level plan to meet the producer's risk point which has an acceptance probability of *at least* 0.95 when the proportions of the 2 types of defectives are equal to 0.05 and 0.06, and for the plan to also meet the consumer's risk point, which has an acceptance probability *at most* 0.1 when the proportions of the 2 types of defectives are equal to 0.14, and 0.18. We can assess whether the sequential plan with cell quotas of $m = 5$ good items and $rn = (2,3)$ defectives meets the given PRP and CRP as follows:

```
> assess.multi(rn=c(2,3), m=5, PRP = c(0.05,0.06, 0.95), CRP = c(0.14,0.18, 0.1),
+   stype="s" )
   3-Level Acceptance Sampling Plan Negative Multinomial:
  Acc. Number: m = 5
  Rej. Number(s):  2 3


  Plan CANNOT meet desired risk point(s):
     type 1 type 2 RP P(accept)  Plan P(accept)  ASN
 PRP 0.05   0.06   0.95          0.95649354      5.5020476
 CRP 0.14   0.18   0.1           0.62784922      5.92261962
```

The output shows that the plan cannot meet both risk points. Although the PRP is satisfied since the actual value for P(accept) is 0.956, which exceeds minimum desired level of 0.95, the value of P(accept) for CRP is 0.628 which is greater than the maximum allowable level of

0.1. For sequential sampling, the output also displays the average sampling number(s) ASN for the risk point(s).

4.5 Finding a Sampling Plan

The find.multi.plan function provides a method to find a plan which will meet the specified risk points *PRP* and *CRP*. For example, the implementation of finding a plan for sequential sampling from a lot of size 100 is:

```
> find.multi.plan(PRP=c(0.06, 0.04, 0.06, 0.8), CRP=c(0.14, 0.16, 0.2, 0.1),
+    N= 100,type = "h", stype="seq")
 The optimal plan is:
$m [1] 7
$rn [1] 2 2 2
$p.PRP [1] 0.8056496
$p.CRP [1] 0.08147094
$ASNp [1] 7.589796
$ASNc [1] 5.510192
```

This shows that, in order to meet both risk points, we should take observations one at a time until we get either 7 good items, or 2 of any type of defective. If the former occurs first then the lot should be accepted; otherwise it is rejected. The average sampling number is 7.6 at the producer's quality level, and 5.5 at the consumer's quality level.

We can also find a plan with the same risk points *PRP*, *CRP*, and lot size as above but for fixed sampling:

```
> find.multi.plan(PRP=c(0.06, 0.04, 0.06, 0.8), CRP=c(0.14, 0.16, 0.2, 0.1),
+    N= 100,type = "h")
 The optimal plan is:
$n [1] 11
$rn [1] 2 2 3
$p.PRP [1] 0.8023994
$p.CRP [1] 0.09043282
```

The above output shows that if we want to meet both risk points, we need a sample of size `n=11`. The lot is rejected if the sample contains at least 2 of either of the first two types of defectives, or at least 3 of the third type of defective. Hence to meet both risk points, the sequential sampling procedure requires on average a smaller sample size (`ASN=7.6`) than the corresponding fixed sample size procedure (`n=11`).

4.6 Calculating Cumulative Probabilities for the Distributions

The following are examples of how to calculate the lower tail cumulative probabilities for the distributions which are provided in the **MFSAS** package.

- Multivariate hypergeometric distribution:

  Suppose that a lot of size $N = 100$ contains products which are classified according to 4 levels of product quality, one of which is good items. If $M = (8, 10, 14)$ is the number of items in the lot which fall into each of the three categories of defectives, and we draw a sample of size $n = 15$ (without replacement) from the lot, then the probability that the sample contains no more than 1 item with the first type of defect, no more than 3 items with the second type of defect, and no more than 4 items with the third type of defect is calculated as follows:

```
> X <- c(1,3,4)
>  n <- 15
>  M <- c(8, 10, 14)
>  N <- 100
>  pr <- pmultihyper(X, n, M, N)
>  pr
[1] 0.599595
```

- Multinomial distribution:

  In the same setting as above, if the sampling is done with replacement, or equivalently if the lot size is large and the proportions of each of the 3 types of defects in the lot is given by $p = (0.08, 0.10, 0.14)$, then the corresponding probability is calculated using the multinomial CDF:

```
> X <- c(1, 3, 4)
> n <- 15
> pr <- c(0.08, 0.10, 0.14)
> cdf <- pmultinom(x = X, size = n, prob = pr)
> cdf
[1] 0.5816256
```

- Negative multivariate hypergeometric distribution:

  Suppose that a lot of size $N = 130$ contains items with 4 different types of product defects with $M = (5, 7, 8, 3)$ of each type. If we select and inspect items one at a time (without replacement) until we obtain 5 good items then the probability that we end up with no more than $X = (2, 3, 4, 1)$ items with each type of defect (at the time when the 5^{th} good item is selected) is calculated as follows:

```
> X <- c(2,3,4,1)

> m <- 5

> M <-c(5,7,8,3)

> N <-130

> cdf <- pnmultihyper(X, m, M, N)

> cdf

[1] 0.990882
```

- Negative multinomial distribution:

  In the same setting as above, if the sampling is done with replacement then the corresponding probability requires the negative multinomial distribution:

```
> X <- c(2,3,4,1)

> m <-5

> pr <-c(5/130, 7/130, 8/130, 3/130)

> pnmultinom(x = X, m = m, prob = pr)

[1] 0.9860325
```

Chapter 5

Discussion and Future Work

The **MFSAS** package provides the user with the tools to create, evaluate, and plot multilevel acceptance sampling plans for both fixed and sequential sampling. We have also provided cumulative distribution functions for several discrete multivariate distributions.

However, the **MFSAS** package is restricted to single stage sampling for attributes, whereas the **AcceptanceSampling** package allows the sampling to be multi-stage fixed sampling plans for two levels of product quality and provides functionality for sampling inspection by variables, in addition to attributes. Multi-stage sampling may be incorporated into future versions of the package.

It should also be noted that all of the procedures used for the calculations in this package are exact (aside from possible rounding errors). As a result there is the potential for the calculations to be slow for large values of `k`, `m`, or `n`. Consequently the `find.multi.plan` routine can be slow if the probability given in the producer's risk point `PRP[k]` is too close to 1 or if `CRP[k]` is too close to 0. Therefore the package could be improved by developing and incorporating asymptotic approximations into the calculations.

In this project, the cumulative distribution functions only include the option for calculating lower tail probabilities. The option to calculate for upper tail probabilities is available for most (if not all) of the currently available R functions for discrete univariate `CDFs`. However, the relationship between upper and lower tail probabilities is not as straight forward in the multivariate case. Efficient calculation of upper tail probabilities would require a different (but related) set of recursive algorithms. As a result of this, and the fact that upper tail probabilities are not needed in this package, our `CDFs` do not currently have this functionality. Since upper tail probabilities may be useful in other applications, this is another area in which the package can be improved.

Appendix A

MFSAS Package Documentation

R objects are documented in files written in "R documentation" (Rd) format, a simple markup language much of which closely resembles the LaTeX, which can be processed into a variety of formats, including LATEX, HTML and plain text.

An 'Rd' file consists of three parts.

- The header, which gives basic information about the name of the file, the topics documented, a title, a short textual description and R usage information for the objects documented (the header is mandatory);
- The body gives further information (for example, on the function's arguments and return value);
- Finally, there is an optional footer with keyword information.

The compiled **MFSAS** package documentation is given on the following pages.

Package 'MFSAS'

June 3, 2010

Type Package

Title Creation and Evaluation of Multilevel Fixed and Sequential Sampling Plans

Version 1.0-1

Date 2010-03-07

Author Aaron Childs and Yalin Chen

Maintainer Aaron Childs `<childsa@mcmaster.ca>`

Description This package provides functionality for creating and evaluating acceptance sampling plans for attributes when there are k (>=2) levels of product quality. Plans can be multilevel fixed, or multilevel sequential.

Depends methods, R(>= 2.9.2), stats

Imports graphics

License GPL (>= 3)

LazyLoad yes

R topics documented:

`assess.multi`	*Utility Function for Assessing Multilevel Sampling Plans*

Description

Assess whether the `k`-level fixed or sequential sampling plan can meet the specified *Producer's Risk Point (PRP)* and/or *Consumer's Risk Point (CRP)*.

Usage

```
assess.multi(rn, n = 30, m, N = 100, PRP, CRP, type=c("multinomial",
 "hypergeom"), stype=c("fixed", "sequential"), print = TRUE)
```

Arguments

`rn`	A vector of length `k-1` of rejection numbers for fixed sampling, or cell quotas for the defective items in sequential sampling.
`n`	Sample size; applicable for `stype="fixed"`.
`m`	The cell quota for good items; applicable for `stype="sequential"`.
`N`	The population (lot) size from which the sample is drawn; applicable for `type="hypergeom"`.
`PRP`	The Producer's Risk Point in the form of a two part numeric vector `(pd, pa)`. The first part `pd`, a vector of length `k-1`, specifies the quality level at which to evaluate the plan. The second part `pa`, indicates the *minimum* probability of acceptance to be achieved by the plan.
`CRP`	The Consumer's Risk Point in the form of a two part numeric vector `(pd, pa)`. The first part `pd`, a vector of length `k-1`, specifies the quality level at which to evaluate the plan. The second part `pa`, indicates the *maximum* probability of acceptance to be achieved by the plan.
`type`	The type of distribution on which the sampling plan is based. Possible values are `"multinomial"` and `"hypergeom"`. The default is `"multinomial"`.
`stype`	The type of sampling. Possible values are `"fixed"` and `"sequential"`. The default is `"fixed"`.
`print`	Logical, indicating whether or not a summary of the assessment should be printed.

Details

Typical usages are:

```
assess.multi(rn, n, PRP, CRP)
assess.multi(rn, m, PRP, CRP, stype ="sequential")
assess.multi(rn, n, N, PRP, CRP, type="hypergeom")
assess.multi(rn, m, N, PRP, CRP, type = "hypergeom", stype="sequential")
```

In the first form, the default `type` `"multinom"` and the default `stype` `"fixed"` are used.

The second form is based on the negative multinomial distribution.

The third form uses a default `stype` of `"fixed"` and is based on the multivariate hypergeometric distribution.

The fourth form is based on the negative multivariate hypergeometric distribution.

The cell quota `m` for the good items must be provided in both second and fourth forms.

In both third and fourth cases, the population size `N` needs to be specified, and `pd` in `PRP` and `CRP` is the vector of the proportions of population defectives. Since `pd*N` gives a vector containing the actual numbers of each type of defective in the population, all of its entries must be integers.

Value

The function will return the result of whether the plan meets the acceptance requirement(s), along with the actual acceptance probability achieved by the sampling plan. In the case of sequential sampling, average sampling numbers `ASNp` and `ASNc` for the quality levels in `PRP` and `CRP` are also returned.

Source

For sequential sampling, the average sampling number `ASN` is computed using algorithms for the Dirichlet `D` function (for `type="multinomial"`) or **HD** function (for `type="hypergeom"`), together with equation (5.30) in Sobel and Frankowski (1985) (for `type="multinomial"`), or equation (5.3) in Childs (2010) (for `type="hypergeom"`).

Childs, A. (2010). Vector extensions of the Dirichlet HC and HD functions, with applications to the sharing problem. *Methodology and Computing in Applied Probability* **12,** 91-109.

Sobel, M. and Frankowski, K. (1985), Dirichlet integrals of type-2 and their application. In *Selected Tables in Mathematical Statistics* **9**, American Mathematical Society, Providence, Rhode Island.

Author(s)

Aaron Childs and Yalin Chen

References

Schilling, E. G. and Neubauer, D. V. (2009). *Acceptance Sampling in Quality Control,* Second Edition, CRC Press, New York.

See Also

`find.multi.plan`, `OCmult-class`

Examples

```
assess.multi(n=30, rn=c(2,2,3), PRP = c(0.05,0.06, 0.08, 0.95),
 CRP = c(0.15,0.18, 0.20, 0.075))
assess.multi(rn=c(7,8), m=5, PRP = c(0.1,0.05, 0.95), CRP = c(0.2,0.15, 0.075),
 type="multinomial", stype="seq")
```

find.multi.plan *Utility Function for Finding Multilevel Sampling Plans*

Description

Find the k-level sampling plan with the smallest (expected) sample size such that the specified *Producer's Risk Point (PRP)* and *Consumer's Risk Point (CRP)* are met.

Usage

```
find.multi.plan(PRP, CRP, N=100, type=c("multinomial", "hypergeom"),
stype=c("fixed", "sequential"))
```

Arguments

PRP	The Producer's Risk Point in the form of a two part numeric vector of the form (pd, pa). The first part pd, a vector of length k-1, specifies the quality level at which to evaluate the plan. The second part pa, indicates the *minimum* probability of acceptance to be achieved by the plan.
CRP	The Consumer's Risk Point in the form of a two part numeric vector of the form (pd, pa). The first part pd, a vector of length k-1, specifies the quality level at which to evaluate the plan. The second part pa, indicates the *maximum* probability of acceptance to be achieved by the plan.
N	The population (lot) size from which the sample is drawn; applicable for type="hypergeom".
type	The distribution on which the sampling plan is based. Possible values are "multinomial" and "hypergeom". The default is "multinomial".
stype	The type of sampling. Possible values are "fixed" and "sequential". The default is "fixed".

Details

Typical usages are:

```
find.multi.plan(PRP, CRP)
find.multi.plan(PRP, CRP, stype ="sequential")
find.multi.plan(PRP, CRP, N, type="hypergeom")
find.multi.plan(PRP, CRP, N, type="hypergeom", stype="sequential")
```

In the first form, the default type "multinomial" and the default stype "fixed" are used.

The second form is based on the negative multinomial distribution.

The third form uses a default stype of "fixed" and is based on the multivariate hypergeometric distribution.

The fourth form is based on the negative multivariate hypergeometric distribution.

In both third and fourth cases, the population size N needs to be specified, and pd in PRP and CRP is the vector of the proportions of population defectives. Since pd*N gives a vector containing the actual numbers of each type of defective in the population, all of its entries must be integers.

Value

The values returned are

`n`	The smallest possible sample size for `stype="fixed"`.
`rn`	Vector of length `k-1` of rejection numbers for fixed sampling, or cell quotas for the defective items in sequential sampling.
`m`	The smallest possible number of good items for `stype="sequential"`.
`p.PRP`	The actual probability of acceptance at the producer's quality level for the sampling plan.
`p.CRP`	The actual probability of acceptance at the consumer's quality level for the sampling plan.
`ASNp`	The average sampling number at the producer's quality level for `stype="sequential"`.
`ASNc`	The average sampling number at the consumer's quality level for `stype="sequential"`.

Source

For sequential sampling, the average sampling numbers `ASNp` and `ASNc` are computed using algorithms for the Dirichlet `D` function (for `type="multinomial"`) or **HD** function (for `type="hypergeom"`), together with equation (5.30) in Sobel, Uppuluri, and Frankowski (1985) (for `type="multinomial"`), or equation (5.3) in Childs (2010) (for `type="hypergeom"`).

Childs, A. (2010). Vector extensions of the Dirichlet HC and HD functions, with applications to the sharing problem. *Methodology and Computing in Applied Probability* **12,** 91 - 109.

Sobel, M., Uppuluri, V. R. R., and Frankowski, K. (1985). Dirichlet integrals of type-2 and their application. In *Selected Tables in Mathematical Statistics* **9**, American Mathematical Society, Providence, Rhode Island.

Author(s)

Aaron Childs and Yalin Chen

References

Schilling, E. G. and Neubauer, D. V. (2009). *Acceptance Sampling in Quality Control*, Second Edition, CRC Press, New York.

See Also

`Ocmult-class`, `Ocmult`, `assess.multi`.

Examples

```
find.multi.plan(PRP=c(0.03, 0.05,  0.8), CRP=c(0.15, 0.16, 0.1), stype="seq")

find.multi.plan(PRP=c(0.06, 0.04, 0.06, 0.8), CRP=c(0.14, 0.16, 0.2, 0.1),
 N= 100,type = "h", stype="seq")
```

`Ocmult`	*Operating Characteristics of Multilevel Acceptance Sampling Plans*

Description

Creating new objects from the `"Ocmult"` classes.

Usage

```
Ocmult(rn, type=c("multinomial", "hypergeom"),
       stype=c("fixed", "sequential"), ...)
```

Arguments

`rn`	A vector of length `k-1` of rejection numbers for `k`-level (`k` different types of items in the population) fixed sampling, or a vector of length `k-1` of cell quotas for the defective items in `k`-level sequential sampling.
`type`	The type of distribution on which the plan is based. Possible values are `"multinomial"` and `"hypergeom"`. The default is `"multinomial"`.
`stype`	The type of sampling. Possible values are `"fixed"` and `"sequential"`. The default is `"fixed"`.
`...`	Additional arguments to be passed to the class generating function for each type. See Details for options.

Details

Typical usages are:

```
Ocmult(rn, n)
Ocmult(rn, n, pd)
Ocmult(rn, n, N, pd, type="hypergeom")
Ocmult(rn, m, stype="sequential", pd)
Ocmult(rn, m, N, pd, type="hypergeom", stype="sequential")
```

In the first and second forms, the default `type` `"multinomial"` and the default `stype` `"fixed"` are used. The OC function is calculated based on the proportion of defectives `pd`, whose default values are used in the first form (and depend on the length of `rn`).

The third form is the OC function based on the multivariate hypergeometric distribution. In this case, the population size `N` needs to be specified, and `pd` is a matrix whose rows are vectors containing the proportions of each type of defective. Since `pd*N` is a matrix containing the actual numbers of each type of defective in the population, all of its entries must be integers. If `N` is not specified, it takes a default value of `N=100`.

The fourth form uses a default `type` of `"multinomial"`. Its OC function is based on the negative multinomial distribution, hence the cell quota `m` for good items must be specified.

In the fifth form, the OC function is based on the negative multivariate hypergeometric distribution. The cell quota `m` for good items and the population size `N` need to be specified.

Value

An object from `Ocmult-class` returns the class `Ocmult.multinomial` or `Ocmult.hypergeom`. There is a logic argument `detail` in the function `summary`. If `detail=TRUE`, all of the information for the object is shown. For sequential sampling the average sampling number (`ASN`) is also provided. The default value for this argument is `detail=FALSE`.

Author(s)

Aaron Childs and Yalin Chen

See Also

`Ocmult-class`, `find.multi.plan`, `assess.multi`.

Examples

```
px <- as.matrix(expand.grid(seq(0,0.5, 0.1),seq(0,0.5, 0.1)))
p.multinom <- Ocmult(n=30,rn=c(3,4), pd=px)
summary(p.multinom ,detail=TRUE )

p.multih<- Ocmult(c(3,4),n=15,  N=100, type="h")
summary(p.multih,detail=TRUE)
persp(p.multih)

p.nmultinom <- Ocmult(c(3,4), m=5, stype="s")
p.nmultinom
summary(p.nmultinom ,detail=TRUE)
persp(p.nmultinom)
```

`Ocmult-class` *Class "Ocmult"*

Description

"Operating Characteristic" function of the class "`Ocmult`" provides methods for creating, plotting and printing `k`-level acceptance sampling plans based on the Multinomial, Negative Multinomial ("`Ocmult.multinomial`"), Multivariate Hypergeometric and Negative Multivariate Hypergeometric ("`Ocmult.hypergeom`") distributions.

Objects from the Class

The "`Ocmult`" class is a virtual class: No objects may be created from it.

However, objects from the derived classes `Ocmult.multinomial` and `Ocmult.hypergeom` can be created using the create function `Ocmult`.

Slots

`n`: Object of class "`numeric`". The sample size; applicable for `stype="fixed"`.

`m`: Object of class "`numeric`". The cell quota for good items; applicable for `stype="sequential"`.

`rn`: Object of class "`numeric`". A vector of length `k-1` of rejection numbers for fixed sampling, or cell quotas for the defective items in sequential sampling.

`pd`: Object of class `"matrix"`. A matrix whose rows are vectors containing the proportions of each type of defective.

`pa`: Object of class `"numeric"`. A numeric vector contains the probabilities of acceptance according to the proportion of defectives in the rows of `pd`.

`ASN`: Object of class `"numeric"`. Only for sequential sampling. A numeric vector containing average sampling numbers according to the proportion of defectives in the rows of `pd`.

`stype`: Object of class `"character"`. The type of sampling. Possible values are `"fixed"` and `"sequential"`.

`type`: Object of class `"character"`. The type of distribution on which the plans are based. Possible values are `"multinomial"` and `"hypergeom"`.

`N`: Object of class `"numeric"`. Only for class `"Ocmult.hypergeom"`. A number giving the population (lot) size from which the sample is drawn.

Methods

show `signature(object = "Ocmult")`: Show the details of the sampling plan.

summary `signature(object = "Ocmult")`: Summarize the sampling plan. Optional argument `"full"` (defaults to `FALSE`) will show the details at all quality values `"pd"` supplied when the object was created.

plot `signature(x = "Ocmult.multinomial", y = "missing")`,
`signature(x = "numeric", y = "Ocmult.multinomial")`,
`signature(x = "Ocmult.hypergeom", y = "missing")`:
`signature(x = "numeric", y = "Ocmult.hypergeom")`:
Plot the OC curve for 2-level sampling plans.

persp `signature(x = "Ocmult.multinomial")`,
`signature(x = "Ocmult.hypergeom")`:
Plot the OC surface for 3-level sampling plans.

contour `signature(x = "Ocmult.multinomial")`,
`signature(x = "Ocmult.hypergeom")`:
Plot the OC contour curve for 3-level sampling plans.

Author(s)

Aaron Childs and Yalin Chen

References

Schilling, E. G. and Neubauer, D. V. (2009). *Acceptance sampling in quality control*, Second Edition, CRC Press, New York.

See Also

`Ocmult`, `find.multi.plan`, `assess.multi`.

`pmultihyper` *The Cumulative Distribution Function for the Multivariate Hypergeometric Distribution*

Description

Compute cumulative probability for the multivariate hypergeometric distribution.

Usage

```
pmultihyper(x, n, M, N)
```

Arguments

`x`	Vector of length `k-1` of non-negative integers where `k` is the number of classes.
`n`	The sample size.
`M`	Vector of length `k-1` containing the total number of objects in each of `k-1` of the classes.
`N`	The size of the population from which the sample is drawn.

Details

The multivariate hypergeometric distribution is used for sampling from a finite population *without* replacement. If a sample of size `n` is drawn from a population of size `N` which has $M[i]$ objects of type i (for $i = 1, 2, ..., k$), let $X[i]$ be the number of objects of type i in the sample (for $i = 1, 2, ..., k$). Then the cumulative probability `pmultihyper(x, n, M, N)` is given by,

$$P(X[1] <= x[1], \ldots, X[k-1] <= x[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \cdots \sum_{y[k-1]=0}^{x[k-1]} P(X[1] = y[1], \ldots, X[k-1] = y[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \cdots \sum_{y[k-1]=0}^{x[k-1]} \frac{\binom{M[1]}{y[1]} \cdots \binom{M[k-1]}{y[k-1]} \binom{N - \sum_{i=1}^{k-1} M[i]}{n - \sum_{i=1}^{k-1} y[i]}}{\binom{N}{n}}$$

where the sum is over all values of y such that $y[1] + y[2] + ... + y[k-1] <= n$ and $n - (y[1] + y[2] + ... + y[k-1]) <= N - (M[1] + M[2] + ... + M[k-1])$.

Value

`pmultihyper` gives the value of the cumulative distribution function. Invalid arguments will stop running.

Source

`pmultihyper` is computed using recursive algorithms for the Dirichlet **HJ** function given in

Sobel, M. and Frankowski, K. (1994). Hypergeometric analogues of multinomial type-1 Dirichlet problems. *Congressus Numerantium* **101**, 65-82.

Author(s)

Aaron Childs and Yalin Chen

References

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1997). *Discrete Multivariate Distributions*, Wiley, New York.

See Also

`pmultinom`, `pnmultinom`, `pnmultihyper`.

Examples

```
X <- c(1,3,4)
n <- 15
M <- c(8, 10, 14)
N <- 50
pr <- pmultihyper(X, n, M, N)
pr
```

`pmultinom` *The Cumulative Distribution Function for the Multinomial Distribution*

Description

Compute cumulative probability for the multinomial distribution.

Usage

```
pmultinom(x, size, prob)
```

Arguments

`x`	Vector of length `k-1` of non-negative integers where `k` is the number of classes.
`size`	The sample size
`prob`	Numeric non-negative vector of length `k-1` specifying the probability for `k-1` of the classes.

Details

The multinomial distribution is used for sampling *with* replacement, or if the population is large compared to the sample size. If a sample of size n is drawn from a population whose k classes have probabilities $p[1], ..., p[k-1], p[k]$, let $X[1], ..., X[k-1], X[k]$ denote the number of observations drawn from each of the k classes. Then the cumulative probability `pmultinom(x, size=n, prob=p)` is given by

$$P(X[1] <= x[1], \ldots, X[k-1] <= x[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \cdots \sum_{y[k-1]=0}^{x[k-1]} P(X[1] = y[1], \ldots, X[k-1] = y[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \cdots \sum_{y[k-1]=0}^{x[k-1]} \frac{n!}{y[1]! \ldots y[k-1]! \left(n - \sum_{i=1}^{k-1} y[i]\right)!} p[1]^{y[1]} \ldots p[k-1]^{y[k-1]}$$

$$* \left(1 - \sum_{i=1}^{k-1} p[i]\right)^{\left(n - \sum_{i=1}^{k-1} y[i]\right)}$$

where the sum is over all values of y such that $y[1] + y[2] + ... + y[k-1] <= n$.

Value

`pmultinom` gives the value of the cumulative distribution function. Invalid arguments will stop running.

Source

`pmultinom` is computed using recursive algorithms for the Dirichlet **J** function given in

Sobel, M., Uppuluri, V. R. R., and Frankowski, K. (1977). Dirichlet distributions type-1. In *Selected Tables in Mathematical Statistics* **4**, American Mathematical Society, Providence, Rhode Island.

Sobel, M. and Frankowski, K. (2004). Extensions of Dirichlet integrals: their computation and probability applications. In *Gupta, A.K. and Nadarajah, S. (eds) Handbook of Beta Distribution and its applications,* 319-360, Marcel Dekker, New York.

Author(s)

Aaron Childs and Yalin Chen

References

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1997). *Discrete Multivariate Distributions*, Wiley, New York.

See Also

`pmultihyper`, `pnmultinom`, `pnmultihyper`

Examples

```
X <- c(2,3)
n <- 20
pr <- c(0.12, 0.15)
cdf <- pmultinom(x = X, size = n, prob = pr)
cdf
```

`pnmultihyper` *The Cumulative Distribution Function for the Negative Multivariate Hypergeometric Distribution*

Description

Compute cumulative probability for the negative multivariate hypergeometric distribution.

Usage

```
pnmultihyper(x, m, M, N)
```

Arguments

`x`	Vector of length `k-1` for the failures where `k` is the number of classes.
`m`	The target number of successful trials. Must be a strictly positive integer.
`M`	Vector of length `k-1` containing the total number of each type of failure in the population.
`N`	Total population size from which the sample is drawn.

Details

The negative multivariate hypergeometric distribution is used for sequential sampling from a finite population *without* replacement. Suppose that the population of size `N` has `k-1` different types of failures represented `M[1], ..., M[k-1]` times. Let `X[1], ..., X[k-1]` denote the number of failures of each type that are selected in a sequence of trials before a target number `m` of successes is reached. Then `pnmultihyper(x, m, M, N)` is the cumulative probability:

$$P(X[1] <= x[1], \dots, X[k-1] <= x[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \dots \sum_{y[k-1]=0}^{x[k-1]} P(X[1] = y[1], \dots, X[k-1] = y[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \dots \sum_{y[k-1]=0}^{x[k-1]} \frac{\binom{M[1]}{y[1]} \cdots \binom{M[k-1]}{y[k-1]} \binom{N - \sum_{i=1}^{k-1} M[i]}{m-1}}{\binom{N}{m-1+\sum_{i=1}^{k-1} y[i]}} \frac{N-m+1-\sum_{i=1}^{k-1} M[i]}{N-m+1-\sum_{i=1}^{k-1} y[i]}$$

Value

`pnmultihyper` gives the value of the cumulative distribution function. Invalid `M` or `N` will stop running with a warning.

Source

`pnmultihyper` is computed using recursive algorithms for the Dirichlet **HD** function given in

Childs, A. (2010). Vector extensions of the Dirichlet HC and HD functions, with applications to the sharing problem. *Methodology and Computing in Applied Probability* **12,** 91-109.

Sobel, M. and Frankowski, K. (1995). Hypergeometric analogues of multinomial type-2 problems via Dirichlet methodology. *Congressus Numerantium* **106**, 171-191.

Author(s)

Aaron Childs and Yalin Chen

References

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1997). *Discrete Multivariate Distributions*, Wiley, New York.

See Also

`pmultinom`, `pmultihyper`, `pnmultinom`.

Examples

```
 X <- c(2,3,4,1)
 m <- 5
 M <-c(5,7,8,3)
 N <-30
cdf <- pnmultihyper(X, m, M, N)
cdf
```

`pnmultinom` *The Cumulative Distribution Function for the Negative Multinomial Distribution*

Description

Compute cumulative probability for the negative multinomial distribution.

Usage

```
pnmultinom(x, m, prob)
```

Arguments

`x`	Vector of length `k-1` for the failures where `k` is the number of classes.
`m`	The target number of successful trials. Must be a strictly positive integer.
`prob`	Numeric non-negative vector of length `k-1` specifying the probability for the `k-1` classes of failures.

Details

The negative multinomial distribution is used for sequential sampling *with* replacement. Suppose that the population has `k-1` different types of failures, with corresponding probabilities `p[1], ..., p[k-1]` in each trial. Let `X[1], ..., X[k-1]` denote the number of failures of each type that are selected in a sequence of trials before a target number `m` of successes is reached. Then `pnmultinom(x, m, prob = p)` is the cumulative probability:

$$P(X[1] <= x[1], \ldots, X[k-1] <= x[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \cdots \sum_{y[k-1]=0}^{x[k-1]} P(X[1] = y[1], \ldots, X[k-1] = y[k-1])$$

$$= \sum_{y[1]=0}^{x[1]} \cdots \sum_{y[k-1]=0}^{x[k-1]} \frac{\left(\sum_{i=1}^{k-1} y[i]+m-1\right)!}{(m-1)!y[1]!\ldots y[k-1]!} p[1]^{y[1]} \ldots p[k-1]^{y[k-1]} \left(1 - \sum_{i=1}^{k-1} p[i]\right)^m$$

Value

`pnmultinom` gives the value of the cumulative distribution function. Invalid `m` or `prob` will stop running with a warning.

Source

`pnmultinom` is computed using recursive algorithms for the Dirichlet **D** function given in

Sobel, M. and Frankowski, K. (1985). Dirichlet integrals of type-2 and their application. In *Selected Tables in Mathematical Statistics* **9**, American Mathematical Society, Providence, Rhode Island.

Sobel, M. and Frankowski, K. (2004). Extensions of Dirichlet integrals: their computation and probability applications. In *Gupta, A. K. and Nadarajah, S. (eds) Handbook of Beta Distribution and its applications,* 319-360, Marcel Dekker, New York.

Author(s)

Aaron Childs and Yalin Chen

References

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1997). *Discrete Multivariate Distributions*, Wiley, New York.

See Also

`pmultinom`, `pmultihyper`, `pnmultihyper`

Examples

```
X <- c(4,5,6)
m <-3
pr <-c(0.10,0.15,0.18)
pnmultinom(x = X, m = m, prob = pr)
```

Index

Appendix B

MFSAS Package Code

The **MFSAS** package code is divided into five parts in following five files: `Ocmult.R`, `functions.R`, `cdf.R`, `assess.R`, and `findplan.R`.

B.1 Ocmult.R

This part uses the `setClass` function to create a virtual class and two actual classes. The function `new` is called to create an object and the `setMethod` function is applied to generate the generic function `show`, `summary`, `plot`, `persp`, and `contour` for the object. This file also contains small functions for the titles of the plots.

```
## Class definitions
## Create a virtual class
setClass("Ocmult",
   representation(
     n="numeric",
       # An integer - the sample size for fixed sampling
     m="numeric",
       # An integer - cell quota of good items for sequential sampling
     rn="numeric",
       # A vector of rejection numbers for fixed sampling or cell quotas
       #of defectives for sequential sampling
```

```
      pd="matrix",
        # The proportions of each type of defective in the population (by row)
      pa="numeric",       # Probability of acceptance
      asn="numeric",      # Average sampling number
      stype="character",  # Type of sampling
      type="character",   # Distribution type
      "VIRTUAL"),
    validity=function(object){
      if(any(is.na(object@rn)))
        return("The 'rn' vector is not allowed to contain missing values.")
      # Check that the rejection numbers are reasonable
      if(any(object@rn < 1))
        return("rejection number(s) 'rn' must be greater than 0.")
      if(any(is.na(object@pd)))
        return("'pd' is not allowed to contain missing values.")
      l=ncol(object@pd)
      # Check that the rows of pd have the same length as rn
      if(length(object@rn)!=l)
        return("The number of columns in 'pd' must be the same as the length of 'rn'.")
      # Check that the proportions of defectives are reasonable
      if(any(object@pd < 0))
        return("The entries in 'pd' must be greater than or equal to 0.")
      if(any(apply(object@pd,1,sum) > 1))
        return("The row sums of 'pd' must be less than or equal to 1.")
      return(TRUE)
    }
)
## Create two classes
setClass("Ocmult.multinomial",
    representation("Ocmult"
                  ),
    prototype=list("Ocmult",
        stype="fixed",
        type="multinomial",
        n=30,
        pd=as.matrix(seq(0, 0.1, by=0.01),nrow=1)
        ),
    contains="Ocmult",
    validity=function(object){
      ## fixed sampling plan
      if(object@stype=="fixed"){
        if(is.na(object@n) )
          return("The 'n' is not allowed to contain missing value.")
        if(length(object@n)!= 1)
          return("The length of sample size 'n' should be equal to 1")
        # Check that the sample size is reasonable
      if(object@n <= 0)
        return("Sample size 'n' should be greater than 0.")
      # Check that the rejection numbers are reasonable
      if(any(object@rn-1 > object@n))
      return("Each rejection number in the vector 'rn' must be less than 'n+1'.")
    }
  # Check that the length of acceptance is reasonable for sequential sampling plan
  if(object@stype=="sequential"){
    if(is.na(object@m) )
      return("The 'm' is not allowed to contain missing value.")
    if(length(object@m)!= 1)
      return("The length of cell quota of good item 'm' should be equal to 1")
    # Check that the cell quota is reasonable
    if(object@m <= 0)
      return("cell quota 'm' should be greater than 0.")
    }
  return(TRUE)
  }
)
setClass("Ocmult.hypergeom",
  representation("Ocmult",
     N="numeric"       # An integer - the population size
     ),
  prototype = list("Ocmult",
      stype="fixed",
      type="hypergeom",
      N=100,
      n=30,
      pd=as.matrix(seq(0, 0.1, by=0.01),nrow=1)
      ),
  contains="Ocmult",
  validity=function(object){
    if(is.na(object@N)) return("N contains NA. ")
    # Check that the population size is reasonable
    if(length(object@N)>1) return("N must be of length 1.")
    if(object@N < 1) return("N is less than 1. ")
    if(length(object@rn) >= object@N)
      return("The length of 'rn' must be less than 'N'.")
    # Check that the numbers of defectives for each type are integers
    if(any((object@N*object@pd < round(object@N*object@pd
        ject@pd > round(object@N*object@pd
        )+1e-6)))
        return("N times pd must be integer numbers.")
    ## fixed sampling plan
    if(object@stype=="fixed"){
      if(is.na(object@n) )
        return("The 'n' is not allowed to contain missing value.")
      if(length(object@n)!= 1)
        return("The length of sample size 'n' should be equal to 1")
      # Check that the sample size is reasonable
      if( object@n <= 0)
        return("Sample size 'n' should be greater than 0.")
      # Check that the rejection numbers are reasonable
      if(any(object@rn-1 > object@n))
```

```
            return("Any rejection number in 'rn' must be less than 'n+1'.")
          # Check that the population size and sample size are reasonable
          if(object@N <= object@n)
            return("N must be greater than n.")
            }
        # Check that the length of acceptance is reasonable for sequential sampling plan
        if(object@stype=="sequential")
          {
          if(is.na(object@m))
            return("The 'm' is not allowed to contain missing value.")
          if(length(object@m)!= 1)
            return("The length of cell quota 'm' should be equal to 1")
          # Check that the cell quota of good item is reasonable
          if( object@m <= 0)
            return("Cell quota 'm' should be greater than 0.")
          # Check that the population size and cell quota are reasonable
          if(object@N <= object@m)
            return("N must be no less than m.")
          }
        return(TRUE)
        }
      )
## Creation of the object
Ocmult <- function(rn, type=c("multinomial", "hypergeom"),
   stype=c("fixed", "sequential"), ...)
   {
   # Choose what 'type' to use
   type <- match.arg(type)        ## Get type, defaut is multinomial
   stype <- match.arg(stype)      ## Get type, defaut is fixed
   # invoke a new object of that type
   switch(type,
     multinomial={
       obj <- new("Ocmult.multinomial", rn=rn, type="multinomial", stype=stype, ...)
     switch(stype,
       fixed={
         obj@pa <- pmultinom(x=obj@rn-1, size=obj@n, prob=obj@pd)
       },
       sequential={
         obj@pa <- calc.pnmultinom(pd=obj@pd, rn=obj@rn, m=obj@m)
         obj@asn <- EWT(pd=obj@pd, rn=obj@rn, m=obj@m)
       }
     )
    },
    hypergeom={
      obj <- new("Ocmult.hypergeom", rn=rn, type="hypergeom", stype=stype, ...)
      switch(stype,
        fixed={
          obj@pa <- calc.pmultihyper(obj@pd, rn=obj@rn, n=obj@n, N=obj@N)
        },
        sequential={
          obj@pa <- calc.pnmultihyper(pd=obj@pd, rn=obj@rn, m=obj@m, N=obj@N)
          obj@asn <- EWTH(pd=obj@pd, rn=obj@rn, m=obj@m, N=obj@N)
        }
      )
    }
  )
  return(obj)
}
##Initialization of the class
setMethod("initialize", "Ocmult.multinomial",
  function(.Object, rn, n=.Object@n, m, pd, stype=.Object@stype, ...)
    {
    lrn <- length(rn)
    if(missing(pd))
      {
      if(lrn==2)
        {
        .Object@pd <- as.matrix(expand.grid(seq(0, 0.1, 0.01), seq(0, 0.1, 0.01)))
        }
      else
      if(lrn > 2)
        {
        if(lrn < 11)
          .Object@pd <- as.matrix(t(combn(seq(0.01, 0.1, 0.01),lrn)))
        else
          {
          lrnpd <- seq(0.1/lrn, 0.1, 0.1/lrn)
          if(sum(lrnpd) < 1)
            .Object@pd <- as.matrix(t(lrnpd))
          else
            .Object@pd <- as.matrix(t(rep(1/(2*lrn), lrn)))
            }
          }
      }
    else if(is.vector(pd))
      {
      if(lrn > 1)
        {
        l.row <- length(pd)/lrn
        if(l.row < 1)
          stop("The length of 'rn' must less than or equal to length of 'pd'.")
        if(l.row > floor(l.row))
          {
          l.row <- floor(l.row)
          l.pd <- lrn*l.row
          pd <- pd[1:l.pd]
          warning(
          paste("The length of the pd vector should be an integer",
                "multiple of the length of the rn vector.",
                "\nThe truncated pd in use is: "), t(paste(pd, " ")))
            }
        }
        if(lrn==2){
          pdtemp <- matrix(pd, ncol=lrn, byrow=TRUE)
        .Object@pd <- as.matrix(expand.grid(unique(sort(pdtemp[, 1])),
          unique(sort(pdtemp[, 2]))))
          }
        else
          .Object@pd <- matrix(pd, ncol=lrn, byrow=TRUE)
          }
        else
        if(lrn==2)
          .Object@pd <- as.matrix(expand.grid(unique(sort(pd[, 1])),
                                 unique(sort(pd[, 2]))))
        else
          .Object@pd <- pd
        if(stype=="fixed")
          .Object@n <- n
        else
          .Object@m <- m
        .Object@rn <- rn
        .Object@stype <- stype
        callNextMethod(.Object, ...)## Return to object
        }
)
setMethod("initialize", "Ocmult.hypergeom",
   function(.Object, rn, n=.Object@n, m, pd, N=.Object@N,
            stype=.Object@stype, ...)
      {
      lrn <- length(rn)
      if(missing(pd))
        {
        if(lrn==2)
          {
           .Object@pd <- as.matrix(expand.grid(seq(0, 0.1, 0.01), seq(0, 0.1, 0.01)))
          }
        else
        if(lrn > 2 )
          {
          if(lrn < 11)
            .Object@pd <- as.matrix(t(combn(seq(0.01, 0.1, 0.01),lrn)))
           else{
             lrnpd <- seq(1/N, lrn/N, 1/N)
             if(sum(lrnpd) < 1)
               .Object@pd <- as.matrix(t(lrnpd))
             else
               .Object@pd <- as.matrix(t(rep(1/N, lrn)))
               }
             }
            }
          else if(is.vector(pd))
            {
            if(lrn > 1)
              {
              l.row <- length(pd)/lrn
              l.row <- length(pd)/lrn
              if(l.row < 1)
                stop("The length of 'rn' must less than or equal to length of 'pd'.")
              if(l.row > floor(l.row))
                {
                l.row <- floor(l.row)
                l.pd <- lrn*l.row
                pd <- pd[1:l.pd]
                warning(
                paste("The length of the pd vector should be an integer",
                      "\nmultiple of the length of the rn vector.",
                      "\nThe truncated pd in use is: \n"), t(paste(pd, " ")))
                }
              }
            if(lrn==2){
              pdtemp <- matrix(pd, ncol=lrn, byrow=TRUE)
              .Object@pd <- as.matrix(expand.grid(unique(sort(pdtemp[, 1])),
                              unique(sort(pdtemp[, 2]))))
            }
            else
              .Object@pd <- matrix(pd, ncol=lrn, byrow=TRUE)
          }
          else
          if(lrn==2)
            .Object@pd <- as.matrix(expand.grid(unique(sort(pd[, 1])),
                              unique(sort(pd[, 2]))))
          else
            .Object@pd <- pd
          if(stype=="fixed")
            .Object@n <- n
          else
            .Object@m <- m
          .Object@rn <- rn
          .Object@N <- N
          .Object@stype <- stype
          callNextMethod(.Object, ...)
          }
        )
## Create show function
setMethod ("show" , "Ocmult",
   function(object)
     {
     if(object@stype=="fixed")
       {
       switch(object@type,
          multinomial=cat(" ", paste(length(object@rn)+1,
             "-Level Acceptance Sampling Plan Multinomial:", sep=""), "\n"),
          hypergeom=cat(" ", paste(length(object@rn)+1,
             "-Level Acceptance Sampling Plan Multivariate Hypergeom: N = ",
            object@N, sep=""), "\n" )
          )
      cat(" Sample size:   ", paste(object@n), "\n" )
      cat(" Rej. Number(s):", paste(object@rn), "\n" )
      cat("\n" )
      }
    else {
      switch(object@type,
         multinomial=cat(" ", paste(length(object@rn)+1,
           "-Level Sequential Acceptance Sampling Plan",
           " Negative Multinomial:", sep=""), "\n"),
         hypergeom=cat(" ", paste(length(object@rn)+1,
            "-Level Sequential Acceptance Sampling Plan",
            " Negative Multivariate Hypergeom: N = ",
            object@N, sep=""), "\n")
         )
      cat(" Acc. Number:   ", paste(object@m), "\n" )
      cat(" Rej. Number(s):", paste(object@rn), "\n" )
      cat("\n" )
    }
  }
)
 ## Create summary function
 setMethod("summary", signature(object="Ocmult"),
    function(object, detail=FALSE)
        {
        if(ncol(object@pd)==1) p.nondef <- 1- object@pd
        else p.nondef <- 1-rowSums(object@pd)
        pa <- round(object@pa,7)
        prop <- cbind(object@pd, p.nondef ,pa)
        l=ncol(object@pd)
        defname = as.vector(sapply(" type",  FUN = paste, (1:l)))
        rownames(prop) <- rep(" ", length(object@pa))
        if(object@stype=="fixed")
          colnames(prop) <- c(defname, " P.nondef", " P(accept)")
        else
          {
          asn <- round(object@asn,7)
          prop <- cbind(prop, asn)
          colnames(prop) <- c(defname, " P.nondef", " P(accept)", "     ASN")
            }
        show(object)
          if(detail){
            cat("\n Detailed acceptance probabilities: \n")
            show(prop)
          }
        if(object@stype=="fixed")
          return(invisible(c(list(n=object@n,  rn=object@rn, P= prop))))
        else
          return(invisible(c(list(m=object@m, rn=object@rn, P= prop))))
        }
    )
 ## Creating a new generic function for plot
 setMethod("plot", c(x = "Ocmult.multinomial", y = "missing"),
    function(x, y, type="o", xlab="Proportion Defective", ylab="P(accept)",
        main = main.2dp(x), ...)
        {
        if(length(x@rn)!=1)
            stop("The plot for 2-Level acceptance sampling plan only")
         plot(x@pd, x@pa, type=type, xlab=xlab, ylab=ylab, main=main,...)
        }
    )
 setMethod("plot", signature(x="numeric", y="Ocmult.multinomial"),
    function(x, y, type="o", ylab="P(accept)", main = main.2dp(y),  ...)
        {
        plot(x, y@pa, type=type, ylab=ylab,  main=main, ...)
        }
    )
setMethod("plot", c(x = "Ocmult.hypergeom", y = "missing"),
   function(x, y, type="p", xlab="Proportion Defective", ylab="P(accept)",
       main = main.2dp(x), ...)
       {
       if(length(x@rn)!=1)
           stop("The plot for 2-Level acceptance sampling plan only")
       plot(x@pd, x@pa, type=type, xlim=xlim, ylab=ylab, main=main,...)
      }
 )

 setMethod("plot", signature(x="numeric", y="Ocmult.hypergeom"),
    function(x, y, type="p", ylab="P(accept)", main = main.2dp(y), ...)
      {
      plot(x, y@pa, type=type, xlim=xlim, ylab=ylab, ylim=ylim, main=main, ...)
      }
 )
 setMethod("persp", c(x = "Ocmult.multinomial"),
   function(x, y,  zlab="P(accept)", xlab="p1", ylab="p2",
       main = main.3dp(x), ticktype = "detailed",...)
     {
     if(length(x@rn)!=2)
       stop("The persp plot is only for 3-Level acceptance sampling.")
     p1 <- unique(sort(x@pd[,1]))
     p2 <- unique(sort(x@pd[,2]))
     pa <- matrix(x@pa, nrow=length(p1))
     ## Create three dimension plot
     persp(p1, p2, pa, theta = 20, phi = 30,d=4, expand=0.5,
       zlab=zlab, xlab=xlab, ylab=ylab, main=main, ticktype = ticktype,...)
    }
)
setMethod("persp", c(x = "Ocmult.hypergeom"),
  function(x, y,  zlab="P(accept)", xlab="p1", ylab="p2",
      main = main.3dp(x), ticktype = "detailed",...)
    {
    if(length(x@rn)!=2)
      stop("The persp plot is only for 3-Level acceptance sampling.")
    p1 <- unique(sort(x@pd[,1]))
    p2 <- unique(sort(x@pd[,2]))
    pa <- matrix(x@pa, nrow=length(p1))
    ## Create three dimension plot
    persp(p1, p2, pa, theta = 20, phi = 30,d=4, expand=0.5,
          zlab=zlab, xlab=xlab, ylab=ylab, main=main,
          ticktype = ticktype,...)
  }
)
## Creating a new generic function for contour plot
setMethod("contour", c(x = "Ocmult.multinomial"),
   function(x, y,  nlevel=8, main = main.3dc(x), xlab="p1", ylab="p2",...)
     {
     if(length(x@rn)!=2)
       stop("The contour plot is only for 3-Level acceptance sampling.")
     p1 <- unique(sort(x@pd[,1]))
     p2 <- unique(sort(x@pd[,2]))
     pa <- matrix(x@pa, nrow=length(p1))
     contour(p1, p2, pa, nlevel=nlevel, main=main, xlab=xlab, ylab=ylab,...)
     }
)
setMethod("contour", c(x = "Ocmult.hypergeom"),
   function(x, y,  nlevel=8,  main = main.3dc(x), xlab="p1", ylab="p2",...)
     {
     if(length(x@rn)!=2)
       stop("The contour plot is only for 3-Level acceptance sampling.")
     p1 <- unique(sort(x@pd[,1]))
     p2 <- unique(sort(x@pd[,2]))
     pa <- matrix(x@pa, nrow=length(p1))
     contour(p1, p2, pa, nlevel=nlevel, main=main, xlab=xlab, ylab=ylab,...)
    }
   )
##The function for plot title
main.3dp <- function(obj)
  {
  if(obj@type=="multinomial")
      {
      if(obj@stype=="fixed")
          return(paste("Multinomial OC Surface with  \nn = ", obj@n,
              ", rn = (", obj@rn[1], ",", obj@rn[2], ")", sep=""))
      else
          return(paste("Negative Multinomial OC Surface with \nm = ",
              obj@m, ", rn = (", obj@rn[1], ",", obj@rn[2],")", sep=""))
        }
    else
        {
        if(obj@stype=="fixed")
          return(paste("Multivariate Hypergeometric OC Surface with \nn = ",
            obj@n, ", N = ", obj@N, ", rn = (",
              obj@rn[1], ",", obj@rn[2],")", sep=""))
         else
           return(paste("Negative Multivariate Hypergeometric OC Surface with \nm = ",
              obj@m, ", N = ", obj@N, ", rn = (", obj@rn[1], ",",
              obj@rn[2],")", sep=""))
         }
    }
  main.3dc <- function(obj)
     {
     if(obj@type=="multinomial")
         {
         if(obj@stype=="fixed")
             return(paste("Multinomial OC Contour with  \nn = ",
             obj@n, ", rn = (", obj@rn[1], ",", obj@rn[2],")", sep=""))
         else
             return(paste("Negative Multinomial OC Contour with \nm = ",
                 obj@m, ", rn = (", obj@rn[1], ",", obj@rn[2],")", sep=""))
         }
     else
         {
          if(obj@stype=="fixed")
            return(paste("Multivariate Hypergeometric OC Contour with \nn = ", obj@n,
                 ", N = ", obj@N, ", rn = (", obj@rn[1], ",", obj@rn[2], ")", sep=""))
          else
            return(paste("Negative Multivariate Hypergeometric OC Contour with \nm = ",
              obj@m, ", N = ", obj@N, ", rn = (", obj@rn[1], ",", obj@rn[2],")", sep=""))
          }
     }
  main.2dp <- function(obj)
    {
    if(obj@type=="multinomial")
        {
        if(obj@stype=="fixed")
           return(paste("Binomial OC Curve with \nn = ",
               obj@n, ", rn = ", obj@rn, sep=""))
    else
        return(paste("Negative Binomial OC Curve with \nm = ",
        obj@m, ", rn = ", obj@rn[1], sep=""))
        }
     else
         {
         if(obj@stype=="fixed")
             return(paste("Hypergeometric OC Curve with \nn = ", obj@n,
            ", N = ", obj@N, ", rn = ", obj@rn, sep=""))
        else
            return(paste("Negative Hypergeometric OC Curve with \nm = ", obj@m,
                ", N = ", obj@N, ", rn = ", obj@rn[1], sep=""))
        }
  }
```

B.2 functions.R

The following are the functions for calculating the cumulative probabilities of the multivariate distributions, and average sampling number using the Dirichlet recursive functions. They include the Dirichlet **J** , **HJ** , **D** , and **HD** functions.

```
## calculate the cumulative distribution function for multinomial
## using recursive algorithms for the Dirichlet J function.
## x is a non-negative integer vector.
## prob is a vector or matrix.
## size is a positive integer.
pmultinom <- function(x, size , prob)
  {
  if(any(x < 0))
    stop("'x' must be non-negative")
  if(size <=0)
    stop("'size' must be greater than 1.")
  if(sum(x) > size)
    stop("Sum of 'x' must be not greater than size.")
  if(any(prob>1)|any(prob<0))
    stop("'prob' is out of range (0,1)")
  b=length(x)
  if(b==1)
    {
    if(length(prob)==1)
      cdf <- JV(x+1, size, prob)
    else
      cdf <-sapply(prob, function(P) JV(x+1, size, P))
    }
  else
   {
   if(is.matrix(prob))
     {
     if(ncol(prob)!=b)
       return(strwrap("The length of 'x' and the number columns
           of 'prob' must be equal."),  width = 60)
```

```
        cdf <-apply(prob, 1, function(P) JV(x+1, size, P))
      }
    else
      {
      if(length(prob)!=b)
        return("'x' and 'prob'  must be equal length vectors.")
      cdf <- JV(x+1, size, prob)
    }
   }
   return(cdf)
  }
 ##Dirichlet J vector function:
 ## JV function is used to check if the input is valid.
 ## RJV is recursive function for Dirichlet J function.
 JV <- function(R, n, P)
  {
  if(sum(P) > 1)
    stop("Sum of 'P' must be not greater than 1.")
  if(any( R<=0 ))return(0)
  if(all(P==0))return(1)
  if(sum(P)==1)return(0)
  R <- R[P!=0]
  P <- P[P!=0]
  b <- length(R)
  jv <- RJV(b, R, n, P)
  return(jv)
  }
 RJV <- function(b, R, n, P)
  {
  if((max(P)== min(P)) & (max(R)== min(R)))
      ## Check if all the values in vector P are the same and if all
      ## the values in vector R are the same
    {
    pj = J(b, 0, R[1], n, P[1])
    return(pj)
  }
  sump=0
  if(R[1]<=0)return(sump)
  else {r1 <- R[1]-1}
  R1 <- R[-1]
  p1 <- P[1]
  P1 <- P[-1]/(1-p1)
  P1[P1>1]=1
  for(i in (0:r1))
    {
    sump <- sump + (choose(n,i))*(p1^i)*((1-p1)^(n-i))*RJV(b-1, R1, n-i, P1)
    }
  return(sump)
}
 ## J function for the same r and same p.
## It is part of Dirichlet J recursive function.
J <- function(b,j,r,n,p)
 {
 if((j*r) > n)return(0)
 if((j*r)==n)
    {
    pj=(factorial(n)/(factorial(r))^j)*(p^n)
    return(pj)
    }
 if(j==b){
   pj <- (factorial(n)/(((factorial(r))^b)*factorial(n-b*r)))*(p^(b*r))*((1-b*p)^(n-b*r))
   return(pj)
   }
   pj <- (1/(n-j*r))*(n*(1-j*p) *J(b,j,r,n-1,p)-r*(b-j)*J(b,j+1,r,n,p))
   return(pj)
     }
## calculate the cumulative distribution function for multivariate hypergeometric
## using recursive algorithms for the Dirichlet HJ function.
## n, N are positive integers. # # x is a vector of non-negative integers.
## M is a vector or matrix of non-negative integers.
pmultihyper <- function(x, n, M, N)   ##what if any x > M ?
  {
  if(any(x < 0))
     stop("'x' must be non-negative")
  if(any(M < 0))
     stop("'M' must be non-negative")
  if(n <=0)
     stop("'n' must be greater than 0.")
  if(sum(x)>n)
     stop("'n' must be no less than the sum of 'x'.")
  if(N<=0)
    stop("'N' must be greater than 0.")
  if(n > N)
    stop("'N' must be no less than than 'n'.")
  b=length(x)
  if(b==1)
    {
    if(length(M)==1)
       cdf <- HJV(b, x+1, n, M, N)
    else
      cdf <-sapply(M, function(y) HJV(b, x+1, n, y, N))
    }
  else
    {
    if(is.matrix(M))
      {
      if(ncol(M)!=b)
         stop(strwrap("The length of 'x' and the
           number columns of 'M' must be equal."),width = 60)
      cdf <-apply(M, 1, function(y) HJV(b, x+1, n, y, N))
      }
    else
      {
      if(length(M)!=b)
         stop("'x' and 'M'  must be equal length vectors.")
      cdf <- HJV(b, x+1, n, M, N)
      }
    }
    return(cdf)
  }
## Dirichlet HJ vector function:
## HJV function is used to check if the input is valid.
## RHJV is recursive function for Dirichlet HJ function.
HJV <- function(b, R, n, M, N)
 {
 if(sum(M)>N)
   stop("'N' must be no less than the sum of 'M'.")
 if(n < 0) return(1)
 if(any( R<=0 ))return(0)
 hjv <- RHJV(b, R, n, M, N)
 return(hjv)
 }

RHJV <- function(b, R, n, M, N)
   {
   if(n >= N)return(0)
   if((max(M)== min(M)) & (max(R)== min(R)))
       ## check if all the values in vector M are the same and
       ## if all the values in vector R are the same
     {
     phj = HJ(b, 0, R[1], n, M[1], N)
     return(phj)
     }
   sump=0
   if(R[1]<1)return(sump)
   else {r1 <- R[1]-1}
   R1 <- R[-1]
   m1 <- M[1]
   M1 <- M[-1]
   for(i in (0:min(m1,r1)))
      {
      sump <- sump + (((choose(m1,i))*
        (choose(N-m1,n-i)))/(choose(N, n)))*RHJV(b-1, R1, n-i, M1, N-m1)
      }
   return(sump)
}
## #HJ function for the same r and same M.
## It is part of Dirichlet HJ recursive function.
HJ <- function(b, j, r, n, M, N)
 {
```

```
 if(n >= N)return(0)
 if(r <= 0)return(0)
 if(r >= M)return(1)
 if((n-b*r) >= (N-b*M))return(0)  #add this
 if((j*r) > n)return(0)
  if((j*r)==n)
     {
     ph <- ((choose(M, r))^j)/(choose(N, n))
     return(ph)
     }
   if(j==b){
     ph <- ((choose(M, r))^b)*(choose((N-b*M), (n-b*r)))/(choose(N, n))
     return(ph)
     }
  ph <- (1/(n-j*r))*(n*(1-((j*(M-r))/(N-n+1)))*HJ(b,j,r,
             n-1,M,N)-r*(b-j)*HJ(b,j+1,r,n,M,N))
  return(ph)
 }
## calculate the cumulative distribution function for negative multinomial
## using recursive algorithms for the Dirichlet D function.
## m is a positive integer.
## x is a vector of positive integers.
## prob is a vector or matrix.
pnmultinom <- function(x, m, prob)
  {
  if(any(x < 0))
    stop("'x' must be non-negative")
  if(m <=0)
    stop("'m' must be greater than 1.")
  if(any(prob>1)|any(prob<0))
     stop("'prob' is out of range (0,1)")
  R=x+1
  b=length(x)
  if(b==1)
     {
     if(length(prob)==1)
       cdf <- DV(b, R, m, prob)
     else
       cdf <-sapply(prob,function(P) DV(b, R, m, P))
     }
  else
    {
    if(is.matrix(prob))
      {
      if(ncol(prob)!=b)
         stop(strwrap("The length of 'x' and the number columns
                           of 'prob' must be equal."), width = 60)
      cdf <-apply(prob, 1, function(P) DV(b, R, m, P))
      }
    else
```

```
      {
      if(length(prob)!=b)
         stop("'x' and 'prob'  must be equal length vectors.")
      cdf <- DV(b, R, m, prob)
      }
    }
    return(cdf)
  }
## Dirichlet D vector function:
## DV function is used to check if the input is valid.
## RDV is recursive function for Dirichlet D function.
DV <- function(b, R, m, P)
 {
 if(sum(P)>1)
   stop("Sum of 'P' must be not great than 1.")
 if(any( R<=0 ))return(0)
 p0 <- 1-sum(P)
 P1 <- P
 dv <- RDV(b, R, m, P1, p0)
 return(dv)
 }
RDV <- function(b, R, m, P1, p0)
 {
 if(p0==0) return(0)
 if(p0==1) return(1)
 if((max(P1)== min(P1)) & (max(R)== min(R)))
     ## check if all the values in vector P are the same and
     ## if all the values in vector R are the same
    {
    a=P1[1]/p0
    rdv = D(b, 0, R[1], m, a)
    return(rdv)
    }
 sump=0
 r1 <- R[1]-1
 R1 <- R[-1]
 p1=P1[1]
 P1 <- P1[-1]
 for(i in 0:r1)
    {
    sump <- sump + choose((m+i-1),i)*((p1/(p0+p1))^i)*((p0/(p0+p1
      )^m)*RDV(b-1, R1, m+i, P1, p0+p1)
    }
 return(sump)
 }
## D function for the same r and same p.
## It is part of Dirichlet D recursive function.
D <- function(b, j, r, m, a)
  {
  if( m <= 0 )return(0)
  if(j==b)
     {
     sump <- (factorial(m+r*b-1)/(((factorial(r))^b)*factorial(m-1))
         )*((1/(1+a*b))^m)*((a/(1+a*b))^(b*r))
     return(sump)
  }
  if(m > r)
    {
    temp=0
    for(i in 1:r)
        {
        temp <- temp + (choose((m-i-1),(r-i))/(a^i))*D(b, j+1, r, m-i, a)
        }
    sump <- (1/choose((m-1), r))*temp
    return(sump)
    }
    sump <- (1/(m+j*r))*(m*(1+j*a)*D(b, j, r, m+1, a)+r*(b-j)*D(b, j+1, r, m, a))
    return(sump)
  }
## calculate the cumulative distribution function for negative multivariate
## hypergeometric using recursive algorithms for the Dirichlet HD function.
## b, m, N are positive integers.
## M, R are vector arguments.
pnmultihyper <- function(x, m, M, N)
   {
   if(any(x < 0))
     stop("'x' must be non-negative")
   if(any(M < 0))
     stop("'M' must be non-negative")
   if( m <= 0 ) stop("'m' must be great than 0.")
   R=x+1
   b=length(x)
   if(b==1)
      {
      if(length(M)==1)
        cdf <- HDV(b, R, m, M, N)
      else
        cdf <-sapply(M,function(y) HDV(b, R, m, y, N))
      }
   else
     {
     if(is.matrix(M))
       {
       if(ncol(M)!=b)
         stop(strwrap("The length of 'x' and the number columns
            of 'M' must be equal."),  width = 60)
       cdf <-apply(M, 1, function(y) HDV(b, R, m, y, N))
       }
     else
       {
        if(length(M)!=b)
           stop("'x' and 'M'  must be equal length vectors.")
        cdf <- HDV(b, R, m, M, N)
      }
    }
    return(cdf)
}
##Dirichlet HD vector function:
## HDV function is used to check if the input is valid.
## RHDV is recursive function for Dirichlet HD function.
HDV <- function(b, R, m, M, N)
  {
  if(any( R<=0 ))return(0)
  hdv <- RHDV(b, R, m, M, N)
  return(hdv)
  }
RHDV <- function(b, R, m, M, N)
   {
   if((max(M)== min(M)) & (max(R)== min(R)))
       ## check if all the values in vector M are the same and
       ## if all the values in vector R are the same
     {
     rhdv = HD(b, 0, R[1], m, M[1], N)
     return(rhdv)
     }
   r1 <- R[1]-1
   R1 <- R[-1]
   m1 <- M[1]
   Mm <- N-sum(M)
   if(Mm <= m)return(0)
   M1 <- M[-1]
   sump=0
   for(i in (0:r1))
     {
     sump <- sump + (choose(m1,i)/((choose((N-sum(M1)), (m+i)))*(m+i))
         )*RHDV(b-1, R1, (m+i), M1, N)
     }
   rhdv <- (choose(Mm, m))*m*sump
   return(rhdv)
}
## HD function for the same r and same M.
## It is part of Dirichlet HD recursive function.
HD <- function(b, j, r, m, M, N)
  {
  if(r > M) return(1)
  if(N <= m+b*r)return(0)
  Mm <- N-b*M
  if(Mm <= m)return(0)
  if(j==b)
    {
```

```
    phd <- (m/(m+b*r))*(((choose(M,r))^b)*(choose(Mm, m))/choose(N, (m+b*r)))
    return(phd)
    }
  if(m > r)
    {
    temp=0
    for(i in 0:(r-1))
      {
      temp <- temp +
          (m/(m+i-r))*(choose(M,i)/choose(Mm, (m+i-r)))*HD(b, j+1, r, m+i-r, M, N)
      }
    phd <- (choose(Mm, m)/choose(M,r))*temp
    return(phd)
  }
  phd <- (1/(m+j*r))*(m*(1+((j*(M-r))/(Mm-m)))
    *HD(b, j, r, m+1, M, N)+ r*(b-j)*HD(b, j+1, r, m, M, N))
  return(phd)
}
```

B.3 cdf.R

This section contains the functions that call the probability functions in the previous section in order to calculate the probabilities of acceptance and expected waiting time (for sequential sampling) of the object.

```
## Calculate the pa (the acceptance probabilities) functions.
## They are used in the initialzation function when creating  the object.
## calc.pmultihyper is used for fixed sampling with multivariate
## hypergeometric distribution.
## calc.pnmultinom is used for sequential sampling with negative
## multinomial distribution.
## calc.pnmultihyper is used for sequential sampling with negative
## multivariate hypergeometric distribution.
calc.pmultihyper <- function(pd, rn, n, N)
  {
  M <- round(N*pd)
  if(((N*pd) < (M-1e-6))||((N*pd) > (M+1e-6)))
    stop("'N' times 'pd' must be integer numbers.")
  pa <- pmultihyper(rn-1, n, M, N)
  return(pa)
}
calc.pnmultinom <- function(pd, rn, m)
   {
```

```
   R <- rn-1
   pa <- pnmultinom(R, m, prob=pd)
   return(pa)
}
calc.pnmultihyper <- function(pd, rn, m, N)
  {
   M=round(N*pd)
   if(((N*pd) < (M-1e-6))||((N*pd) > (M+1e-6)))
      stop("'N' times 'pd' must be integer numbers.")
   R=rn-1
   pa <- pnmultihyper(R, m, M, N)
   return(pa)
  }
## Calculate the expected sample size (average sampling number) using
## the D function for the arguments matching the ones in the classes.
## pd is a vector or matix.
## rn is a nonnegtive vector.
EWT <- function(pd, rn, m)
   {
   b=length(rn)
   R=rn
   if(is.matrix(pd))
      {
      if(ncol(pd)!=b)
        return(strwrap("The number of columns in pd must be equal to
          the length of rn.", width = 60))
      ewt <- apply(pd, 1, function(x) EWTD(b, R, m, x))
      }
   else
     {
     if(length(pd)!=b)
       return(strwrap("The length of pd must be equal to the
         length of rn.", width = 60))
     ewt <- EWTD(b, R, m, pd)
     }
   return(ewt)
}
## Calculate the expected sample size (average sampling number) using
## the HD function for the arguments matching the ones in the classes.
## pd is a vector or matix.
## rn is a nonnegtive vector
EWTH <- function(pd, rn, m, N)
    {
    b=length(rn)
    R=rn
    if(is.matrix(pd))
      {
      if(ncol(pd)!=b)
        return(strwrap("The number of columns in pd must be equal to
           the length of rn.", width = 60))
      M <- round(pd*N)
    ewt <- apply(M, 1, function(x) EWTHD(b, R, m, x, N))
    }
    else
       {
       if(length(pd)!=b)
          return(strwrap("The length of pd must be equal to the
             length of rn.", width = 60))
       M <- round(pd*N)
       ewt <- EWTHD(b, R, m, M, N)
       }
    return(ewt)
}
## Calculate the expected sample size (average sampling number) using
## the D function for the arguments defined as Dirichlet D function.
## b, m are positive integers.
## R, P are vector arguments.
EWTD <- function(b, R, m, P)
   {
   if(length(P)!= b | length(R)!= b)
      stop("The length of 'P1'and 'R' must be equal to 'b'.")
   if(any(P>1)|any(P<0))stop("'p' is out of range (0,1)")
   if(sum(P)>1)stop("Sum of 'P' must not be greater than 1.")
   if(any( R<=0 ))return(0)
   p0 <- 1-sum(P)
   if(all(P==0)) return(m)
   else
     {
     RR <- c(m, R)
     PP <- c(p0, P)
     R <- RR[PP!=0]
     P <- PP[PP!=0]
     b <- length(R)
      EPWT <- 0
      for(i in 1:b)
        {
        ri <- R[i]
        Ri <- R[-i]
        pi <- P[i]
        Pi <- P[-i]
        EPWT <- EPWT+(ri/pi)*RDV(b-1, Ri, ri+1, Pi, pi)
        }
      }
     return(EPWT)
   }
## Calculate the expected sample size (average sampling number) using
## the HD function for the arguments defined as Dirichlet HD function.
## b, m, N are positive integers.
## R, M are vector arguments.
EWTHD <- function(b, R, m, M, N)
```

```
   {
   if( m <= 0 ) stop("'m' must be greater than 0.")
   if(length(M)!= b | length(R)!= b)
      stop("The length of 'M'and 'R' must be equal to 'b'.")
   EPWT <- 0
   Mg <- N-sum(M)
   Rm <- c(m, R)
   MM <- c(Mg, M)
   for(i in 1:(b+1))
      {
      ri <- Rm[i]
      Ri <- Rm[-i]
      mi <- MM[i]
      Mi <- MM[-i]
      EPWT <-EPWT+(ri*(N+1)/(mi+1))*RHDV(b, Ri, ri+1, Mi, N+1)
      }
   return(EPWT)
}
```

B.4 assess.R

The following code is for assessing whether the given plan can meet the criteria specified in *PRP(Producer Risk Point) and/or CRP (Consumer Risk Pint).*

```
## Assessment function is used to assess whether the given plan can meet the criteria
## specified in PRP(Producer Risk Point) and/or CRP (Consumer Risk Pint).
assess.multi <- function(rn, n=30, m, N=100, PRP, CRP, type=c("multinomial", "hypergeom"),
   stype=c("fixed", "sequential"), print=TRUE)
   {
   if(any(rn < 1))
      stop("The values in 'rn' must not be less than 1.")
   if(missing(PRP) & missing(CRP))
     stop("At least one risk point, PRP or CRP, must be specified")
   type <- match.arg(type)
   stype <- match.arg(stype)
   if(stype=="sequential"){
     if(missing(m))
       stop("'m' is missing.")
     if(m < 1)
        stop("'m' must be greater than 0.")
     }
   if(missing(PRP))
     {
     l <- length(CRP)
```

```
if(any(CRP < 0)|any(CRP > 1))
   stop("'CRP' is out of range (0,1).")
if(l>2)
    {
    if(sum(CRP[-l])>1)
       stop("Sum of risk point must not be greater than 1.")
    }
PRP <- rep(NA,l)
if(stype=="fixed")
  {
  if(type=="multinomial")
    pcons <- pmultinom(rn-1, n, CRP[-l])
  else
    pcons <- calc.pmultihyper(CRP[-l], rn, n, N)
  }
else
  {
  if(type=="multinomial")
    {
    pcons <- calc.pnmultinom(CRP[-l], rn, m)
    asncons <- EWT(CRP[-l], rn, m)
    }
  else
    {
    pcons <- calc.pnmultihyper(CRP[-l], rn, m, N)
    asncons <- EWTH(CRP[-l], rn, m, N)
    }
  }
  if(stype=="fixed")
    result2 <- c(CRP, pcons )
  else
    result2 <- c(CRP, pcons, asncons)
  if(pcons >= CRP[l])
    plan.meet <- FALSE
  else
    plan.meet <- TRUE
  result <- as.matrix(t(result2))
  defname = as.vector(sapply("type",  FUN = paste, (1:(l-1))))
  if(stype=="fixed")
    dimnames(result) <- list("CRP",
                       c(defname,
                       "RP P(accept) ", "Plan P(accept)"))
  else
    dimnames(result) <- list("CRP",
                     c(defname, "RP P(accept) ",
                     "Plan P(accept) ", "ASN"))
   }
  else if(missing(CRP))
    {
    l <- length(PRP)
    if(any(PRP < 0)| any(PRP[l]>1))
       stop("'PRP' is out of range (0,1).")
    if(l>2)
       {
       if(sum(PRP[-l])>1)
           stop("Sum of risk point must not be greater than 1.")
       }
    CRP <- rep(NA,l)
    if(stype=="fixed")
       {
       if(type=="multinomial")
          pprod <- pmultinom(rn-1, n, PRP[-l])
       else
         pprod <- calc.pmultihyper(PRP[-l], rn, n, N)
       }
    else
      {
      if(type=="multinomial")
        {
        pprod <- calc.pnmultinom(PRP[-l], rn, m)
        asnprod <- EWT(PRP[-l], rn, m)
        }
      else
        {
        pprod <- calc.pnmultihyper(PRP[-l], rn, m, N)
        asnprod <- EWTH(PRP[-l], rn, m, N)
        }
      }
    if(stype=="fixed")
      result1 <- c(PRP, pprod)
    else
      result1 <- c(PRP, pprod, asnprod)
    if(pprod <= PRP[l])
      plan.meet <- FALSE
    else
      plan.meet <- TRUE
    result <- as.matrix(t(result1))
    defname = as.vector(sapply("type",  FUN = paste, (1:(l-1))))
    if(stype=="fixed")
      dimnames(result) <- list("PRP",
                              c(defname,
                             "RP P(accept) ", "Plan P(accept)"))
    else
      dimnames(result) <- list("PRP",
                               c(defname, "RP P(accept) ",
                               "Plan P(accept) ", "ASN"))
  }
else
  {
  l <- length(PRP)
```

```
if(any(CRP[-1] <= PRP[-1]))
  stop( "Consumer Risk Point quality must be greater than Producer
      Risk Point quality.")
if(any(PRP < 0)| any(PRP > 1))
  stop("'PRP' is out of range (0,1).")
if(any(CRP < 0)|any( CRP > 1))
  stop("'CRP' is out of range (0,1).")
if(l>2){if(sum(PRP[-1])>1|sum(CRP[-1])>1)
  stop("Sum of risk point must not be greater than 1.")}
if(stype=="fixed")
  {
  if(type=="multinomial")
    {
    pprod <- pmultinom(rn-1, n, PRP[-1])
    pcons <- pmultinom(rn-1, n, CRP[-1])
    }
  else
    {
    pprod <- calc.pmultihyper(PRP[-1], rn, n, N)
    pcons <- calc.pmultihyper(CRP[-1], rn, n, N)
    }
  }
else
  {
  if(type=="multinomial")
    {
    pprod <- calc.pnmultinom(PRP[-1], rn, m)
    pcons <- calc.pnmultinom(CRP[-1], rn, m)
    asnprod <- EWT(PRP[-1], rn, m)
    asncons <- EWT(CRP[-1], rn, m)
    }
  else
    {
    pprod <- calc.pnmultihyper(PRP[-1], rn, m, N)
    pcons <- calc.pnmultihyper(CRP[-1], rn, m, N)
    asnprod <- EWTH(PRP[-1], rn, m, N)
    asncons <- EWTH(CRP[-1], rn, m, N)
   }
  }
  if(stype=="fixed")
    {
    result1 <- c(PRP, pprod)
    result2 <- c(CRP, pcons )
    }
  else
    {
    result1 <- c(PRP, pprod, asnprod)
    result2 <- c(CRP, pcons, asncons)
    }
  if(pprod <= PRP[1]| pcons >= CRP[1])
```

```
      plan.meet <- FALSE
    else
      plan.meet <- TRUE
    result <- rbind(result1, result2)
    defname = as.vector(sapply("type",  FUN = paste, (1:(l-1))))
    if(stype=="fixed")
      dimnames(result) <- list(c("PRP", "CRP" ),
                               c(defname,
                               "RP P(accept) ", "Plan P(accept)"))
    else
      dimnames(result) <- list(c("PRP", "CRP" ),
                              c(defname, "RP P(accept) ",
                               "Plan P(accept) ", "ASN"))
    }
  if(print)
    {
    if(stype=="fixed")
      {
      if(type=="multinomial")
        {
        cat(" ", paste(l,
          "-Level Acceptance Sampling Plan Multinomial:", sep=""), "\n")
        cat(" Sample size: ", paste(n), "\n" )
        cat(" Rej. Number(s): ", paste(rn),  "\n" )
        cat("\n" )
        }
      else
        {
        cat(" ", paste(l,
          "-Level Acceptance Sampling Plan Multivariate Hypergeometric: N =",
           N, sep=""), "\n")
        cat(" Sample size: ", paste(n), "\n" )
        cat(" Rej. Number(s): ", paste(rn),  "\n" )
        cat("\n" )
      }
    }
  else
    {
    if(type=="multinomial")
      {
      cat(" ", paste(l,
        "-Level Acceptance Sampling Plan Negative Multinomial:", sep=""), "\n")
      cat(" Acc. Number: m =", paste(m), "\n" )
      cat(" Rej. Number(s): ", paste(rn),  "\n" )
      cat("\n" )
      }
    else
      {
      cat(" ", paste(l,
        "-Level Acceptance Sampling Plan Negative Multivariate Hypergeometric: N =",
              N, sep=""), "\n")
           cat(" Acc. Number: m = ", paste(m), "\n" )
           cat(" Rej. Number(s): ", paste(rn),  "\n" )
           cat("\n" )
         }
       }
     if(plan.meet)
       {
       cat(" Plan CAN meet desired risk point(s): \n")}
     else
       {
       cat(" Plan CANNOT meet desired risk point(s): \n")
       }
     print(formatC(result, digits = 8, format = "f", drop0trailing=TRUE),quote = FALSE)
       }
     if(stype=="fixed")
       {
       if(type=="multinomial")
         {
         return(invisible(c(list(n=n, rn=rn,
              result=result, plan.meet=plan.meet))))
         }
       else
         {
         return(invisible(c(list(N=N, n=n, rn=rn,
             result=result, plan.meet=plan.meet))))
         }
       }
     else
       {
       if(type=="multinomial")
         {
         return(invisible(c(list(rn=rn, m=m,
         result=result, plan.meet=plan.meet))))
       }
     else
       {
       return(invisible(c(list(N=N, rn=rn, m=m,
            result=result, plan.meet=plan.meet))))
     }
   }
}
```

B.5 findplan.R

The following code is to find the best plan that meets the criteria specified in *PRP(Producer Risk Point) and CRP (Consumer Risk Pint).*

```
find.multi.plan <- function(PRP, CRP, N=100, type= c("multinomial", "hypergeom"),
    stype= c("fixed", "sequential"))
    {
    type <- match.arg(type)
    stype <- match.arg(stype)
    l=length(PRP)
    k=l-1
    done=0
    if(stype=="fixed")
      {
      if(type=="multinomial")
        {
        n <- 0
        while(done==0)
          {
          n=n+1
          rn=rep(1, k)
           pcons <- pmultinom(rn-1, n, CRP[-l])
           if(pcons <= CRP[l])
             {
              pprod <- pmultinom(rn-1, n, PRP[-l])
              if(pprod >= PRP[l])
                {
                done=1
                break
                }
              else
               {
               func=n.plan(k, rn, n, PRP, CRP, pcons)
               done=func$done
               if(done==1)
                 {
                 pprod=func$pprod
                 pcons=func$pcons
                 rn=func$rn
                 }
               }
              }
             }
           }
          else
            {
```

```
    n <- 0
    while(done==0)
      {
      n=n+1
      rn=rep(1, k)
      pcons <- calc.pmultihyper(CRP[-1], rn, n, N)
      if(pcons <= CRP[1])
        {
        pprod <- calc.pmultihyper(PRP[-1], rn, n, N)
        if(pprod >= PRP[1])
          {
          done=1
          break
          }
        else
          {
          func=h.plan(k, rn, n, N, PRP, CRP, pcons)
          done=func$done
          if(done==1)
            {
            pprod=func$pprod
            pcons=func$pcons
            rn=func$rn
            }
          }
        }
      }
    }
  }
else
 {
 if(type=="multinomial")
   {
   m <- 0
   while(done==0)
     {
     m=m+1
     rn=rep(1, k)
     pprod <- calc.pnmultinom(PRP[-1], rn, m)
     pcons <- calc.pnmultinom(CRP[-1], rn, m)
     if(pcons <= CRP[1] & pprod >= PRP[1])
       {
       done=1
       break
       }
     else
       {
       if(pcons <= CRP[1])
         {
         func=nm.plan(k, rn, m, PRP, CRP, pcons)
```

```
          done=func$done
          if(done==1)
            {
            pprod=func$pprod
            pcons=func$pcons
            rn=func$rn
            m=func$m
            }
          }
        }
      }
      asnprod <- EWT(PRP[-1], rn, m)
      asncons <- EWT(CRP[-1], rn, m)
    }
  else
    {
    m <- 0
    while(done==0)
      {
      m=m+1
      rn=rep(1, k)
      pcons <- calc.pnmultihyper(CRP[-1], rn, m, N)
      if(pcons <= CRP[1])
        {
        pprod <- calc.pnmultihyper(PRP[-1], rn, m, N)
        if( pprod >= PRP[1])
          {
           done=1
           break
          }
        else
          {
          func=nh.plan(k, rn, m, PRP, CRP, N, pcons)
          done=func$done
          if(done==1)
            {
            pprod=func$pprod
            pcons=func$pcons
            rn=func$rn
            m=func$m
            }
          }
        }
      }
      asnprod <- EWTH(PRP[-1],  func$rn, func$m, N)
      asncons <- EWTH(CRP[-1],  func$rn, func$m, N)
    }
  }
if(stype=="fixed")
  result <- list(n=func$n, rn=func$rn, p.PRP=func$pprod, p.CRP=func$pcons)
```

```
      else
         result <- list(m=m, rn=rn, p.PRP=pprod, p.CRP=pcons,
                        ASNp=asnprod, ASNc=asncons)
      cat(" The optimal plan is: \n")
      show(result)
      return(invisible(result))
      }
## recursive find plan function for multinomial distribution
n.plan <- function(b, rn, n, PRP, CRP, pcons)
    {
    l=length(PRP)
    k=l-1
    done=0
    if(b==1)
      {
      while(pcons <= CRP[1])
         {
         pprod <- pmultinom(rn-1, n, PRP[-1])
         pcons <- pmultinom(rn-1, n, CRP[-1])
         if(pcons <= CRP[1] & pprod >= PRP[1])
           {
           done=1
           break
           }
         if(sum(rn-1)>= n)
           break
         else
           rn[k]=rn[k]+1
         }
         if(done==1)
            return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
         else
           {
           if(k!=b)
             {
              rn[k]=1
              rn[k-1]=rn[k-1]+1
              pprod <- pmultinom(rn-1, n, PRP[-1])
              pcons <- pmultinom(rn-1, n, CRP[-1])
             }
           }
         return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
         }
      else
        while(pcons <= CRP[1])
          {
          fuc=n.plan(b-1, rn, n, PRP, CRP, pcons)
          done = fuc$done
          pprod=fuc$pprod
          pcons=fuc$pcons
          if(done==1)
            {
            rn=fuc$rn
            break
            }
          else
            {
            rn[k-b+1] <-rn[k-b+1]+1
            rn[(k-b+2):k] <- 1
            if(pcons <= CRP[1] & pprod >= PRP[1])
              {
              done=1
              break
              }
            }
          if(sum(rn-1)>= n)
            break
          }
        if(done==1)
          return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
        else
          {
          if(k!=b)
            {
            rn[(k-b+1):k] <-1
            rn[k-b]=rn[k-b]+1
            pprod <- pmultinom(rn-1, n, PRP[-1])
            pcons <- pmultinom(rn-1, n, CRP[-1])
            }
          return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
          }
    }

## recursive find plan function for multivariate hypergeometric distribution
h.plan <- function(b, rn, n, N, PRP, CRP, pcons)
  {
  l=length(PRP)
  k=l-1
  done=0
  if(b==1)
    {
    while(pcons <= CRP[1])
      {
      pprod <- calc.pmultihyper(PRP[-1], rn, n, N)
      pcons <- calc.pmultihyper(CRP[-1], rn, n, N)
      if(pcons <= CRP[1] & pprod >= PRP[1])
        {
        done=1
        break
        }
      if(sum(rn-1)>= n)
        break
      else
        rn[k]=rn[k]+1
      }
    if(done==1)
      return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
    else
      {
      if(k!=b)
        {
        rn[k]=1
        rn[k-1]=rn[k-1]+1
        pprod <- calc.pmultihyper(PRP[-1], rn, n, N)
        pcons <- calc.pmultihyper(CRP[-1], rn, n, N)
        }
      }
      return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
    }
  else
    while(pcons <= CRP[1])
      {
      fuc=fuc=h.plan(b-1, rn, n, N, PRP, CRP, pcons)
      done = fuc$done
      pprod=fuc$pprod
      pcons=fuc$pcons
      if(done==1)
        {
        rn=fuc$rn
        break
        }
      else
        {
        rn[k-b+1] <-rn[k-b+1]+1
        rn[(k-b+2):k] <- 1
        if(pcons <= CRP[1] & pprod >= PRP[1])
          {
          done=1
          break
          }
        }
      if(sum(rn-1)>= n)
        break
      }
    if(done==1)
      return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
    else
      {
      if(k!=b)
        {
          rn[(k-b+1):k] <-1
          rn[k-b]=rn[k-b]+1
          pprod <- calc.pmultihyper(PRP[-l], rn, n, N)
          pcons <- calc.pmultihyper(CRP[-l], rn, n, N)
          }
        return(list(n=n, rn=rn, pprod=pprod, pcons=pcons, done=done))
      }
    }
## recursive find plan function for negative multinomial distribution
 nm.plan <- function(b, rn, m, PRP, CRP, pcons)
    {
    l=length(PRP)
    k=l-1
    done=0
    if(b==1)
      {
      while(pcons <= CRP[l])
        {
        pprod <- calc.pnmultinom(PRP[-l], rn, m)
        pcons <- calc.pnmultinom(CRP[-l], rn, m)
        if(pcons <= CRP[l] & pprod >= PRP[l])
          {
          done=1
          break
          }
        if(rn[k] > m)
          break
        else
          rn[k]=rn[k]+1
        }
      if(done==1)
        return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
      else
        {
        if(k!=b)
          {
          rn[k]=1
          rn[k-1]=rn[k-1]+1
          pprod <- calc.pnmultinom(PRP[-l], rn, m)
          pcons <- calc.pnmultinom(CRP[-l], rn, m)
          }
        }
      return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
      }
    else
      while(pcons <= CRP[l])
        {
        fuc=nm.plan(b-1, rn, m, PRP, CRP, pcons)
        done = fuc$done
        pprod=fuc$pprod
      pcons=fuc$pcons
      if(done==1)
        {
        rn=fuc$rn
        break
        }
      else
        {
        rn[k-b+1] <-rn[k-b+1]+1
        rn[(k-b+2):k] <- 1
        if(pcons <= CRP[1] & pprod >= PRP[1])
          {
          done=1
          break
          }
        }
      if(rn[k-b+1] > m)
        break
      }
    if(done==1)
      return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
    else
      {
      if(k!=b)
        {
        rn[(k-b+1):k] <-1
        rn[k-b]=rn[k-b]+1
        pprod <- calc.pnmultinom(PRP[-1], rn, m)
        pcons <- calc.pnmultinom(CRP[-1], rn, m)
        }
      return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
    }
  }

## recursive find plan function for negative multivariate hypergeometric distribution
nh.plan <- function(b, rn, m, PRP, CRP, N, pcons)
  {
  l=length(PRP)
  k=l-1
  done=0
  if(b==1)
    {
    while(pcons <= CRP[1])
      {
      pprod <- calc.pnmultihyper(PRP[-1], rn, m, N)
      pcons <- calc.pnmultihyper(CRP[-1], rn, m, N)
      if(pcons <= CRP[1] & pprod >= PRP[1])
        {
        done=1
        break
        }
        if(rn[k] > m)
          break
        else
          rn[k]=rn[k]+1
        }
      if(done==1)
        return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
      else
        {
        if(k!=b)
          {
          rn[k]=1
          rn[k-1]=rn[k-1]+1
          pprod <- calc.pnmultihyper(PRP[-1], rn, m, N)
          pcons <- calc.pnmultihyper(CRP[-1], rn, m, N)
          }
        }
      return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
    }
  else
    while(pcons <= CRP[1])
      {
      fuc=nh.plan(b-1, rn, m, PRP, CRP, N, pcons)
      pprod=fuc$pprod
      pcons=fuc$pcons
      done = fuc$done
      if(done==1)
        {
        rn=fuc$rn
        break
        }
      else
        {
        rn[k-b+1] <-rn[k-b+1]+1
        rn[(k-b+2):k] <- 1
        if(pcons <= CRP[1] & pprod >= PRP[1])
          {
          done=1
          break
          }
        }
      if(rn[k-b+1] > m)
        break
      }
    if(done==1)
      return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
    else
      {
      if(k!=b)
```

```
        {
        rn[(k-b+1):k] <- 1
        rn[k-b]=rn[k-b]+1
        pprod <- calc.pnmultihyper(PRP[-1], rn, m, N)
        pcons <- calc.pnmultihyper(CRP[-1], rn, m, N)
        }
      return(list(m=m, rn=rn, pprod=pprod, pcons=pcons, done=done))
    }
  }
```

B.6 Testing.R

The following code contains the functions for calculating the cumulative probabilities for the multivariate distributions using multiple summation [equation (2.1) - (2.4)], and the functions for calculating the expected waiting time for sequential sampling using the Dirichlet **J**-functions and **HJ**-functions [equation (2.28) and (2.31)]. These functions, which are excluded in the package, were used to check the corresponding functions used in the package.

```
# Function for calculating proberbility for multnomial distribution
# ac is vector of acceptance number (ac = rn - 1)
pmultbinom <- function(pd, n, ac)
   {
   if(any(pd>1)|any(pd<0))return("'p' is out of range (0,1)")
   if(any(ac>n)|any(ac<0))return("'ac' is out of range (0,n)")
   # expand ac(a vector of acceptance number)
   x=c(1,ac+1)
   l<-length(x)
   y=NULL
   for(i in 2:l)
     {
     y=cbind(y,rep(0:(x[i]-1), each=prod(x[-(2:i)]),
     times=prod(x[(1:(i-1))])))
     }
   # remove defective number exceed sample size
   X <- y[rowSums(y)<= n, ]
   X <- cbind(X, n-rowSums(X))
   if(is.matrix(pd))
     {
     p <- cbind(pd, 1-rowSums(pd))
     pa=NULL
     for(k in c(1:dim(p)[1]))
       {
        pa[k]=sum(apply(X, 1, function(x) dmultinom(x, prob =p[k, ])))
        }
       }
     else
      {
      p <- c(pd, 1-sum(pd))
      pa = sum(apply(X, 1, function(x) dmultinom(x, prob =p)))
     }
   return(pa)
 }
## Function for calculating the density for multivariate hypergeometric distribution
 dmultihyper <- function(x, M)
   {
   #if(any(M < x))return(" 'x' must be less than 'M'.")
   prod(choose(M, x))/choose(sum(M), sum(x))
     }
## Function for calculating the probability for multivariate hypergeometric distribution
 pmultihyper <- function(pd, N, n, ac)
   {
   if(any(pd>1)|any(pd<0))return("'p' is out of range (0,1)")
   if(any(ac>n)|any(ac<0))return("'ac' is out of range (0,n)")
   if(N < n)return(" 'n' must be less than 'N'.")
   # expand ac(a vector of acceptance number)
   x=c(1,ac+1)
   l<-length(x)
   y=NULL
   for(i in 2:l)
     {
     y=cbind(y,rep(0:(x[i]-1), each=prod(x[-(2:i)]),
       times=prod(x[(1:(i-1))])))
     }
   # remove defective number exceed sample size
   X <- y[rowSums(y)<=n, ]
   X <- cbind(X, n-rowSums(X))
   # get a matrix of type defective numbers in population
   MN <- N*pd
   if(is.matrix(pd))
     {
     MN <- cbind(MN, N-rowSums(MN))
     pa=NULL
     #print(dim(X))
     #print(l)
     for(k in c(1:dim(MN)[1]))
       {
       # remove any type defective number exceed the population size
       flag <-apply(X, 1 ,function(x){return(all(x <= MN[k, ]))})
       xn <- X[flag, ]
       if(is.matrix(xn)){
       pa[k]=sum(apply(xn, 1, function(x) dmultihyper (x, M = MN[k, ])))
        }
        else {
          pa[k]= dmultihyper (xn, M = MN[k, ])
          }
        }
      }
    else
      {
      MN <- c(MN, N-sum(MN))
      # remove any type defective number exceed the population size
      flag <- apply(X,1,function(x){return(all(x <= MN)) })
      xn <- X[flag, ]
      pa=sum(apply(xn, 1, function(x) dmultihyper (x, M = MN)))
      }
    return(pa)
}
#calculate the pdf for negative multinomial distribution
#x is a veactor of the number of failures
#m is a target number of successes
#prob is a vector of probabilities include last successe probability in vector
dnmultinom <- function (x, m, prob, log = FALSE)
  {
   K <- length(prob)-1
   if (length(x) != K)
     stop("1+ length of vector 'x' must be equal length vector 'prob'.")
   if (any(prob < 0) || (s <- sum(prob)) == 0)
     stop("probabilities cannot be negative nor all 0.")
   if (s > 1)
     stop("probabilities cannot be greater than 1.")
   y <- x
   x <- c(x, m)
   if (any(x < 0))
     stop("'x','m' must be non-negative")
   x <- as.integer(x + 0.5)
   N <- sum(x)
   i0 <- prob == 0
   if (any(i0))
     {
     if (any(x[i0] != 0))
       return(if (log) -Inf else 0)
     x <- x[!i0]
     prob <- prob[!i0]
     }
   r <- lgamma(N) + sum(x * log(prob)) - sum(lgamma(y + 1)) - lgamma(m)
   if (log)
     r
   else exp(r)
}
#calculate the cdf for negative multinomial distribution
#pd is a marix or a vector of probabilities of failures
#m is ac is a veactor of a target number of successes and the numbers of failures
```

```
 pnmultinom <- function(pd, ac)
   {
   if(any(pd>1)|any(pd<0))
     stop("'p' is out of range (0,1).")
     m=ac[1]
     # expand ac(a vector of acceptance number)
     x=c(1,ac[-1]+1)
     l<-length(x)
     y=NULL
     for(i in 2:l)
       {
       y=cbind(y,rep(0:(x[i]-1), each=prod(x[-(2:i)]),
         times=prod(x[(1:(i-1))]))) #repeat numbers
     ##from 0 to x[i]-1, each number repeat the mutiplication of x[-(2:i)] times,
     ##repeat the mutiplication of x[(1:(i-1))] times.
       }
     #calculate the probability for matix(pd)
     if(is.matrix(pd))
       {
       p <- cbind(pd, 1-rowSums(pd))
       pa=NULL
       for(k in c(1:nrow(p)))
         {
         pa[k]=sum(apply(y, 1, function(x) dnmultinom(x, m, prob =p[k, ])))
         }
       }
     else
     #calculate the probability for vector(pd)
       {
       p <- c(pd, 1-sum(pd))
       pa = sum(apply(y, 1, function(x) dnmultinom(x, m, prob =p)))
       }
     return(pa)
 }
# calculate the density for negative multivariate hypergeometric distribution
 dnmultihyper <- function(x, m, M, N)
   {
   if(m > (N-sum(M)))
     return(0)
   else
     pmf <- ((choose((N-sum(M)), (m-1))*prod(choose(M,x)))/choose(N,(m+sum(x)-1))
       )*((N-sum(M)-m+1)/(N-m-sum(x)+1))
     return(pmf)
   }
# calculate the probability cdf for negative hypergeometric distribution
 pnmultihyper <- function(pd, ac,  N)
   {
   if(any(pd>1)|any(pd<0))
     stop("'p' is out of range (0,1)")
   if(any(ac>N)|any(ac<0))
```

```
  stop("'ac' is out of range (0, N)")
m=ac[1]
# expand ac(a vector of acceptance number)
x=c(1,ac[-1]+1)
l<-length(x)
y=NULL
for(i in 2:l)
  {
  y=cbind(y,rep(0:(x[i]-1), each=prod(x[-(2:i)]), times=prod(x[(1:(i-1))])))
  #repeat numbers from 0 to x[i]-1, each number repeat the multiplication of
  # x[-(2:i)] times, repeat the multiplication of x[(1:(i-1))] times.
  }
# remove defective number exceed population size
 X <- y[(rowSums(y)+m)<N, ]
# get a matrix of type defective numbers in population
MN <- round(N*pd)
#calculate the probability for matix(pd)
if(is.matrix(pd))
  {
  #MN <- cbind(MN, N-rowSums(MN))
  pa=NULL
  for(k in c(1:nrow(MN)))
    {
    #remove any type defective number exceed the population size
    flag <-apply(X,1,function(x){return(all(x <= MN[k, ]))})
    xn <- X[flag, ]
    if(is.matrix(xn))
      {
      pa[k]=sum(apply(xn, 1, function(x) dnmultihyper (x, m, M = MN[k, ],N)))
      }
    else
      {
      pa[k]= dnmultihyper (xn,m, M = MN[k, ],N)
      }
    }
  }
else
#calculate the probability for vector(pd)
  {
  # MN <- c(MN, N-sum(MN))
  #remove any type defective number exceed the population size
  flag <- apply(X,1,function(x){return(all(x <= MN)) })
   xn <- X[flag, ]
   if(is.matrix(xn))
     {
     pa=sum(apply(xn, 1, function(x) dnmultihyper (x,m, M = MN, N)))
     }
   else
     {
     pa= dnmultihyper (xn, m, M = MN, N)
```

```
        }
      }
    return(pa)
  }
# calculate the expected sample size using J function
# b, m are positive integers # # R, P are vector arguments
EWTJ <- function(b, R, m, P)
  {
  if(length(P)!= b | length(R)!= b)
    stop("The length of 'P1'and 'R' must be equal to 'b'.")
    if(any(P>1)|any(P<0))stop("'p' is out of range (0,1)")
    if(sum(P)>1)return("Sum of 'P' must be not great than 1.")
    if(any( R<=0 ))return(0)
    p0 <- 1-sum(P)
    P0 <- P/(1-p0)
    wtg <- 0
    for(i in m:(m+sum(R)-b))
      {
      wtg <- wtg + i*(choose((i-1), (m-1)))*(p0^m)*((1-p0)^(i-m))*RJV(b, R, i-m, P0)
      }
    wtd <- 0
    for(i in 1:b)
      {
      ri <- R[i]
      Ri <- c(m, R[-i])
      pi <- P[i]
      Pi <- c(p0, P[-i])
      PI <- Pi/(1-pi)
      wtdi <- 0
      for(j in ri:(m+sum(R)-b))
        {
        wtdi <- wtdi
          + j*(choose((j-1),(ri-1)))*(pi^ri)*((1-pi)^(j-ri))*RJV(b, Ri, j-ri, PI)
        }
      wtd <- wtd + wtdi
    }
  EPWT <- wtg + wtd
  return(EPWT)
}
# calculate the expected sample size using HJ function
# b, m, N are positive integers
# R M are vector arguments
ETWHJ <- function(b, R, m, M, N)
  {
  if( m <= 0 ) stop("'m' must be great than 0.")
  if(length(M)!= b | length(R)!= b)
    stop("The length of 'M'and 'R' must be equal to 'b'.")
  if((N-sum(M)) < m )
    stop("'m' must be less or equal than the number of good item in the population.")
  if(any( R<=0 ))return(0)
  sm <- sum(M)
  sr <- sum(R)
  EWTG <- 0
  for(i in (m :min(N,(sm+m),(m+sr-b))))
    {
    EWTG <- EWTG + i*(choose((N-sm),(m-1))*choose(sm, (i-m))
      /choose(N, (i-1)))*((N-sm-m+1)/(N-i+1))*RHJV(b, R, i-m, M, sm)
    }
  EWTD <- 0
  MG <- N-sm
  for(i in 1:b)
   {
   mi <- M[i]
   Mi <- c(MG, M[-i])
   ri <- R[i]
   Ri <- c(m, R[-i])
   for(j in ( ri : min(N, N-mi+ri, m+sr-b)))
     {
     EWTD <- EWTD + j*(choose(mi, (ri-1))*choose((N-mi), (j-ri))
     /choose(N, (j-1)))*((mi-ri+1)/(N-j+1))*RHJV(b, Ri, j-ri, Mi, N-mi)
     }
   }
  EPWT <- EWTG + EWTD
  return(EPWT)
}
```

Bibliography

ASA (1950). *Acceptance Sampling - A Symposium.* American Statistical Association, Washington, DC.

Bowker AH, Goode HP (1952). *Sampling Inspection by Variables.* McGraw-Hill, New York.

Bray D, Lyon D, Burr I (1973). "Three class attributes plans in acceptance sampling." *Technometrics*, **15**(3), 575–585.

Cassady CR, Nachlas AJ (2003). "Evaluating and implementing 3-level acceptance sampling plans." *Quality Engineering*, **15**(3), 361–369.

Childs A (2010). "Vector extensions of the Dirichlet HC and HD functions, with applications to the sharing problem." *Methodology and Computing in Applied Probability*, **12**, 91 – 109.

Dodge HF (1943). "A sampling plan for continuous production." *Annals of Mathematical Statistics*, **14**(3), 264–279.

Dodge HF (1969a-c; 1970a). "Notes on the evolution of acceptance sampling plans." *Journal of Quality Technology*, pp. Part I **1**(2) 77–88; Part II **1**(3) 155–162; Part III **1**(4) 225–232; Part IV **2**(1) 1–8.

Dodge HF, Romig HG (1941). "Single sampling and double sampling inspection tables." *The Bell System Technical Journal*, **20**(1), 1–61.

Jennett WJ, Welch BL (1939). "The control of proportion defective as judged by a single quality characteristic varying on a continuous scale." *Supplement to the Journal of the Royal Statistical Society*, **6**, 80–88.

Johnson NL, Kotz S, Balakrishnan N (1997). *Discrete Multivariate Distributions.* Wiley Series in Probability and Statistics. Wiley, New York.

Kiermeier A (2008). "Visualizing and Assessing Acceptance Sampling Plans: The R Package AcceptanceSampling." *Journal of Statistical Software*, **26**(6). URL `http://www.jstatsoft.org/v26/i06/`.

MIL-STD-105E (1989). *Sampling procedures and tables for inspection by attributes.* Department of defense, Washington, DC.

Newcombe P, Allen O (1988). "A three-class procedure for acceptance sampling by variables." *Technometrics*, **30**(4), 415–421.

Pearson ES (1935). *The Application of Statistical Methods to Industrial Standardization and Quality Control.* British Standard 600:1935, British Standards Institution, London.

Romig HG (1939). *Allowable average in sampling inspection.* Ph.D. thesis, Columbia University, New York.

Schilling EG, Neubauer DV (2009). *Acceptance Sampling in Quality Control.* CRC Press, New York, second edition.

Shapiro S, Zaheda H (1990). "Bernoulli trials and discrete distributions." *Journal of Quality Technology*, **22**(3), 193–205.

Sobel M, Frankowski K (1994). "Hypergeometric analogues of multinomial type-1 Dirichlet problems." *Congressus Numerantium*, **101**, 65–82.

Sobel M, Frankowski K (1995). "Hypergeometric analogues of multinomial type-2 problems via Dirichlet methodology." *Congressus Numerantium*, **106**, 171–191.

Sobel M, Frankowski K (2004). *Handbook of Beta Distribution and its applications*, chapter Extensions of Dirichlet integrals: their computation and probability applications, pp. 319–360. Marcel Dekker, New York.

Sobel M, Uppuluri VRR, Frankowski K (1977). *Dirichlet integrals of type-1 and their application*, volume 4 of *Selected Tables in Mathematical Statistics.* American Mathematical Society, Providence, Rhode Island.

Sobel M, Uppuluri VRR, Frankowski K (1985). *Dirichlet integrals of type-2 and their application*, volume 9 of *Selected Tables in Mathematical Statistics.* American Mathematical Society, Providence, Rhode Island.

Thatcher FS, Clarke DS (1978). "Micro-organisms in food 2: Sampling for Microbiological Analysis, Principles and Specific Applications." *University of Toronto Press.*