Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Friedman test: a C program for nonparametric two-way analysis of variance and multiple comparisons on ranks

Wenbi Wang

Walter R. Dyck

Ghee W. Ho

Defence R&D Canada – Toronto
Technical Memorandum
DRDC Toronto TM 2004-214
November 2004

Canada

Friedman test: a C program for nonparametric two-way analysis of variance and multiple comparisons on ranks

Wenbi Wang

Walter R. Dyck

Ghee W. Ho

Defence R&D Canada – Toronto

Technical Memorandum

DRDC Toronto TM 2004-214

November 2004

Author

Ghee W. Ho and Denis P.C. Tang

Approved by

Keith Hendy

Head, Human Factors Research and Engineering Section

Approved for release by

K.M. Sutton

Chair, Document Review and Library Committee

This page intentionally left blank.

Table of contents

Acknowledgements

The author would like to thank Dr. Ghee W. Ho and Mr. Walter R. Dyck for their contribution in this project. They initially identified the lack of program tools for conducting the Friedman, Quade, and relevant post-hoc tests. They conducted a thorough investigation among commercial statistical analyzing software packages (e.g., SAS, SPSS, STATISTICA). It was their findings from that exercise which motivated the start of this project.

This page intentionally left blank.

Background and Motivation

Ordinal data gathered from repeated administration of questionnaires, i.e., repeated measures, employing a rating scale are commonly deployed in field and laboratory studies. If there are a large number of subjects (e.g., n>30), the assumptions of parametric approach, namely, normality and homogeneity of variance are usually met. Therefore, parametric analysis of variance (ANOVA) methods are frequently adopted to analyze these data.

However, in field and laboratory trials conducted at DRDC Toronto, situations have been frequently encountered in which small number of subjects, e.g., n < 15, are tested in repeated measure experiments. Due to the relatively small sample size, the violation of assumptions of an ANOVA is usually inevitable. Under these situations, researchers should use the Friedman test instead. The Friedman test is a nonparametric two-way ANOVA method that compares two or more related samples, and is equivalent to the repeated measures or within-subjects design of classical ANOVA procedures, where the assigned order of treatments is done randomly for each subject [1-3].

In our investigation, we found that although commercial statistical packages such as SPSS, SAS, and STATISTICA can perform a Friedman ANOVA [4-6], they lack the capability to easily and accurately carry out a post-hoc multiple comparison procedure, after Friedman test scores show a significant difference between the treatment groups.

Having seen a need for an accessible post-hoc multiple comparison procedure, the second and third authors of this report, implemented a Friedman test algorithm originally published by [7] using Quick BASIC, a programming language supported by the DOS Operating System. This program automates the Friedman score calculation and significantly simplifies the post-hoc multiple comparison analysis. Since then, this Quick Basic program (hereafter referred as *the BASIC program*) has been successfully applied in several DRDC projects, for example, CF CADPAT Rain suit evaluation trial, CF converged rain suit evaluation trial, CF snow boots (Mukluk) trial [8-10]. However, due to the constraints imposed by the programming language itself and the stringent requirement on its operating environment, this BASIC program has several intrinsic usability problems which create unnecessary difficulties to use and consequently limit its potentials. The major problems that have been identified are summarized as below.

1. Inadequate I/O. One of the major complaints received from users is the lack of I/O module. This BASIC program requires the raw data for analysis to be specified in a matrix format, stored in a 2D array, and defined inside the source code. As a result,

each time when a new set of data is to be analyzed, a programmer is needed to open the source code and populate the data manually. Additionally, two internal variables (i.e., row, col) are used in the program to keep track of the size of the data matrices. Values of these two variables also need to be assigned manually before the execution of the program. Currently, the program does not have a mechanism to check whether the assigned row and column values match the actual data size. This has been found to be particularly problematic. A typical error is that a programmer updated the raw data set, but forgot to change the values of the row and column variables. Without any warning, the BASIC program will continue to compute the test scores which are obviously mistaken.

Outputs of the program are the Friedman test scores. For verification purposes, the original raw data set and the intermediate computation results are also printed out for review. Currently, the BASIC program outputs all these information directly onto the screen. Because the program is running only under MS-DOS environment, it becomes a problem when the length of an output is longer than a screen view. The BASIC program does not provide a way for the user to retrieve the information beyond the current screen view. One awkward solution is suggested to work around this problem by reducing the size of the raw data set and ensures that the output can be printed inside one screen view.

2. Operating system constraints. The program was written in Quick BASIC on DOS. Since the Quick BASIC is an *interpreted* language, in contrast to a *compiled* language such as C/C++, the running of a BASIC program requires an interpreter which reads in one line of a program and executes it before going to the next line. There are two primary drawbacks with this interpretation approach. Firstly, the interpreter needed for executing this BASIC program only runs on DOS. Consequently, to use the BASIC program for a Friedman test, one needs to find a computer which speaks DOS and has a Quick BASIC installed on it. Since the primary Operating System currently installed and supported at DRDC Toronto is the MS-Windows family OSs, it increasingly becomes a headache to locate a DOS machine for running the BASIC program. Secondly, another problem with the interpretation approach is its slow execution. Since the interpreter must read a line, translate it, find the corresponding machine level code, and then execute the instructions, interpreted language program runs significantly slower than its compiled language counterpart. This imposes a potential limitation of the BASIC program especially when there is a large data set is analyzed.

3. Lack of feedback and help. Feedback during program execution often serves as diagnostic information that allows users to interpret the state of the analysis, while the help functions provide additional support for using a program. Those are important usability features. However, because Quick BASIC supports very limited interaction features, the BASIC program does not produces any feedback during its execution and does not provide any help functions.

4. Poor result formatting. After a Friedman test revealed a significant treatment effect, a post-hoc pair-wise comparison is conducted. Results of each comparison are printed individually. Users need to refer back to the mean rank value of each treatment before categorizing different treatment conditions and draw meaningful conclusions.

These problems together determine that the BASIC program will only be a tool that a few can manage to use. It is our intention to revamp the program, enhance it with user-friendly features, and improve its interfaces so that more researchers will benefit from it.

The C program

Considering the C language is widely supported by most Operating Systems, it was decided that the new Friedman test was rewritten using C. This new program (hereafter referred as *the C program*) consists of several modules. The core of the C program is the Friedman test score calculation. The same algorithm used in the previous BASIC program was adopted and directly translated into C. An input module was created which allow the program to read in raw data from a data file and automatically calculate the size of the data matrices. Compared with the output from the BASIC program, the C program enhanced its output module. Results from pair-wise comparison were further analyzed and categorized following the style used in other Statistics software, e.g., SAS. Analysis results were stored in a text file and could be reviewed by any text editor. The source code of the full C program is included in the annex of this report. The enhanced features of the C program are highlighted here.

1. Multiple OS support. The C program is ANSI C compliant, which ensures that the executables generated from this code can be compiled and run on different operating systems. Currently, the code has passed tests on Linux (Redhat 8.0, 9.0) and Ms-Windows (98, ME, 2000, XP) environments. Since the executables are compiled from the same source code, interface consistency is ensured between the Windows and Linux versions of the program.

2. Separating data from algorithm. Raw data is stored in a separate text file. Before running an analysis, one needs to specify the name/location of the raw data file and the C program will read all the data into the memory, automatically recognize the size (i.e., row and column numbers) of the raw data set, and compute the Friedman scores.

3. Constant feedback. While the program is performing the calculation, text feedback is print out on the screen to inform users the current state of calculation.

4. Structured output. Friedman test scores, together with the intermediate computation results are stored in an output file for reviewing. A complete capturing of the computation process allows users to identify and locate potential problems that might happen during the program execution. A SAS style significance plot is also generated for ease interpretation of the results.

5. A help function is included for clear command line instructions.

6. The compiled program provides fast execution performance compared with the original BASIC program.

7. To further simplify the data file specification, a batch script is created for Windows version of the C program. The user can specify the data source file inside the script and the script will automatically call up the executables, process the source data, compute the Friedman scores, and show the analysis results by bring up the default text editor defined by the operating systems.

Instructions on how to use the C program

Installation

Useful program files are compressed into a single zip file, named *friedman.zip*. This zip file is self-extractable which means that no uncompressing software is further needed to extract program files. Double clicking the friedman.zip file icon will initiate the installation. Following on-screen instructions, users are able to specify default directory and go through the installation.

Files

A new directory called */friedman* is created after the installation. This is the default working directory for running the program. Initially, it includes three files and two sub-directories. They are explained in detail below.

1. *friedman.bat* This is a script batch file in which users specify source data for analysis. Double clicking on this file will start an analysis. In sequence, this batch script will first call friedman.exe, compute the Friedman scores, and then call the default text editor to reveal the results.

2. *sample_data.txt* Raw data need to be stored in a text file and using a comma delimited format. For demonstration purposes, a sample data file is included.

3. *friedman_output.txt* This is the output file where test results are stored. It can be reviewed by any text editor (e.g., a notepad). Detailed explanation of a typical output will be presented later on in this report.

4. */exe/friedman.exe* Sub-directory */exe* include the main executable of the C program, *friedman.exe*. This executable contains the core algorithm for perform the data analysis. Users do not need to interact directly with this program. It is called upon by *friedman.bat* during an analysis.

5. */src/friedman.c* This */src* sub-directory holds a copy of the C source code. Permission is granted to anyone to use this software for research purpose, and to alter it and redistribute it freely.

Data analysis process

Conducting a Friedman test using the C program typically involves three steps:

1. Format raw data
All data should be formatted as *comma delimited* and saved into a standalone text file inside the default working directory, i.e., */friedman*. For demonstration purpose, a sample data file can be found in the default working directory i.e., */friedman/sample_data.txt*.

2. Specify raw data inside the batch script *friedman.bat*.
The *friedman.bat* can be opened with any text editor (e.g., notepad.exe). For ease explanation, the content of a batch script is included here. The only user modification is required on line 13, where the actual source data file, as created in step #1, needs to be specified. The rest of the content is self-explanatory.

```
1.  echo off
2.  rem # Script for running Friedman and Quade tests under Windows OSs
3.  rem #
4.  rem # Created by: Wenbi.Wang@drdc-rddc.gc.ca
5.  rem # Last modified: July 16, 2004
6.
7.  rem ###########################################################################
8.  rem # INSTRUCTIONS: replace the "sample_data.txt" with the real data file
9.  rem ###########################################################################
10.
11. echo on
12.
13. call \exe\friedman.exe sample_data.txt
14. call write.exe friedman_output.txt
```

3. Initiate the analysis by double-clicking on the batch script.
Upon completion, output file is then automatically saved and further presented for reviewing.

Output break down

Output from an analysis is stored in a text file, *friedman_output.txt*, and stored in the default working directory, */friedman.* A typical output can be divided into two sections. For ease interpretation, a sample output file is included in the report.

Section 1 displays the raw data and three intermediate computation results. The first line of the output file shows the source data file that is used for analysis. In this example, it is *sample_data.txt.* Then the size of the source data, row and column

number of the data matrix, and the full set of raw data are displayed. User can double check the data in the output file to ensure that the raw data were correctly read into the memory for processing. Following the raw data, three intermediate calculation results, i.e., rank of sample ranges, sum of ranks for each treatment, and intermediate variable $S_{ij,}$ are presented for debugging purposes.

Section 2 shows the Friedman test results, the p value. In situation a significant treatment effect has been identified, further pair-wise comparison is performed and results are included, as shown in the included example output. Treatment conditions are then categorized and displaced based on three significance levels.

```
================================================================================
Source data file for Friedman test: sample_data.txt
================================================================================
The size of the raw data matrix is: row = 12; column = 8.

 A list of your raw data:

  -10.000    13.000    42.000    28.000    41.000    31.000     9.000     0.000
    2.000     6.000   210.000   398.000   235.000   198.000    99.000    73.000
    6.000    35.000   403.000   270.000   251.000   117.000    44.000    21.000
  -13.000   -29.000   344.000   260.000   161.000   177.000   -79.000   -81.000
    0.000     0.000   729.000   579.000   596.000   386.000   318.000   300.000
    0.000    51.000    47.000    27.000     1.000    19.000    -3.000    -1.000
    0.000     6.000    18.000    -3.000   -24.000    10.000    -9.000     1.000
    1.000    17.000    50.000    65.000    34.000    -1.000   -28.000    -1.000
   17.000   159.000    72.000    50.000    28.000    34.000    37.000    -1.000
    9.000    80.000   148.000   146.000    84.000    -1.000    64.000    35.000
   29.000    29.000   226.000   298.000   148.000   137.000    92.000    71.000
   17.000   -19.000    71.000   166.000    78.000   -24.000     8.000    -8.000

A list of your ranked data:

Rank of sample ranges:

    1.000      4.000      8.000      5.000      7.000      6.000      3.000      2.000
        sample range =   52.000,       rank of sample range:  2
    1.000      2.000      6.000      8.000      7.000      5.000      4.000      3.000
        sample range =  396.000,       rank of sample range:  9
    1.000      3.000      8.000      7.000      6.000      5.000      4.000      2.000
        sample range =  397.000,       rank of sample range: 10
    4.000      3.000      8.000      7.000      5.000      6.000      2.000      1.000
        sample range =  425.000,       rank of sample range: 11
    1.500      1.500      8.000      6.000      7.000      5.000      4.000      3.000
        sample range =  729.000,       rank of sample range: 12
    3.000      8.000      7.000      6.000      4.000      5.000      1.000      2.000
        sample range =   54.000,       rank of sample range:  3
    4.000      6.000      8.000      3.000      1.000      7.000      2.000      5.000
        sample range =   42.000,       rank of sample range:  1
    4.000      5.000      7.000      8.000      6.000      2.500      1.000      2.500
        sample range =   93.000,       rank of sample range:  4
    2.000      8.000      7.000      6.000      3.000      4.000      5.000      1.000
        sample range =  160.000,       rank of sample range:  6
    2.000      5.000      8.000      7.000      6.000      1.000      4.000      3.000
        sample range =  149.000,       rank of sample range:  5
    1.500      1.500      7.000      8.000      6.000      5.000      4.000      3.000
        sample range =  269.000,       rank of sample range:  8
    5.000      2.000      6.000      8.000      7.000      1.000      4.000      3.000
        sample range =  190.000,       rank of sample range:  7

Sum of ranks for each treatment:
```

```
   30.000    49.000    88.000    79.000    65.000    52.500    38.000    30.500

List of Sij:

   -7.000    -1.000     7.000     1.000     5.000     3.000    -3.000    -5.000
  -31.500   -22.500    13.500    31.500    22.500     4.500    -4.500   -13.500
  -35.000   -15.000    35.000    25.000    15.000     5.000    -5.000   -25.000
   -5.500   -16.500    38.500    27.500     5.500    16.500   -27.500   -38.500
  -36.000   -36.000    42.000    18.000    30.000     6.000    -6.000   -18.000
   -4.500    10.500     7.500     4.500    -1.500     1.500   -10.500    -7.500
   -0.500     1.500     3.500    -1.500    -3.500     2.500    -2.500     0.500
   -2.000     2.000    10.000    14.000     6.000    -8.000   -14.000    -8.000
  -15.000    21.000    15.000     9.000    -9.000    -3.000     3.000   -21.000
  -12.500     2.500    17.500    12.500     7.500   -17.500    -2.500    -7.500
  -24.000   -24.000    20.000    28.000    12.000     4.000    -4.000   -12.000
    3.500   -17.500    10.500    24.500    17.500   -24.500    -3.500   -10.500

*******************************
Results from the Friedman test
The Friedman test A2 = 2446.500
The Friedman test B2 = 2220.125
the Friedman test T2 =   13.417        p = 8.300820e-07

Difference between sums for Friedman test:

Observation 1
    -observation 1 =    0.000   NS
    -observation 2 =   19.000   p<0.05
    -observation 3 =   58.000   p<0.001
    -observation 4 =   49.000   p<0.001
    -observation 5 =   35.000   p<0.001
    -observation 6 =   22.500   p<0.01
    -observation 7 =    8.000   NS
    -observation 8 =    0.500   NS
Observation 2
    -observation 1 =   19.000   p<0.05
    -observation 2 =    0.000   NS
    -observation 3 =   39.000   p<0.001
    -observation 4 =   30.000   p<0.001
    -observation 5 =   16.000   p<0.05
    -observation 6 =    3.500   NS
    -observation 7 =   11.000   NS
    -observation 8 =   18.500   p<0.05
Observation 3
    -observation 1 =   58.000   p<0.001
    -observation 2 =   39.000   p<0.001
    -observation 3 =    0.000   NS
    -observation 4 =    9.000   NS
    -observation 5 =   23.000   p<0.01
    -observation 6 =   35.500   p<0.001
    -observation 7 =   50.000   p<0.001
    -observation 8 =   57.500   p<0.001
Observation 4
    -observation 1 =   49.000   p<0.001
    -observation 2 =   30.000   p<0.001
    -observation 3 =    9.000   NS
    -observation 4 =    0.000   NS
    -observation 5 =   14.000   p<0.05
    -observation 6 =   26.500   p<0.01
    -observation 7 =   41.000   p<0.001
    -observation 8 =   48.500   p<0.001
Observation 5
    -observation 1 =   35.000   p<0.001
    -observation 2 =   16.000   p<0.05
    -observation 3 =   23.000   p<0.01
    -observation 4 =   14.000   p<0.05
```

```
    -observation 5 =    0.000   NS
    -observation 6 =   12.500   NS
    -observation 7 =   27.000   p<0.001
    -observation 8 =   34.500   p<0.001
Observation 6
    -observation 1 =   22.500   p<0.01
    -observation 2 =    3.500   NS
    -observation 3 =   35.500   p<0.001
    -observation 4 =   26.500   p<0.01
    -observation 5 =   12.500   NS
    -observation 6 =    0.000   NS
    -observation 7 =   14.500   p<0.05
    -observation 8 =   22.000   p<0.01
Observation 7
    -observation 1 =    8.000   NS
    -observation 2 =   11.000   NS
    -observation 3 =   50.000   p<0.001
    -observation 4 =   41.000   p<0.001
    -observation 5 =   27.000   p<0.001
    -observation 6 =   14.500   p<0.05
    -observation 7 =    0.000   NS
    -observation 8 =    7.500   NS
Observation 8
    -observation 1 =    0.500   NS
    -observation 2 =   18.500   p<0.05
    -observation 3 =   57.500   p<0.001
    -observation 4 =   48.500   p<0.001
    -observation 5 =   34.500   p<0.001
    -observation 6 =   22.000   p<0.01
    -observation 7 =    7.500   NS
    -observation 8 =    0.000   NS

Means with the same grouping letters are NOT significantly different!
======================================================
Significance level: 0.001
       Grouping              Treatment      Mean

                      A          1          30.000000

                      A          8          30.500000

                      A          7          38.000000

            B         A          2          49.000000

         C  B         A          6          52.500000

      D  C  B                    5          65.000000

      D  C                       4          79.000000

      D                          3          88.000000

======================================================
Significance level: 0.01
       Grouping              Treatment      Mean

                      A          1          30.000000

                      A          8          30.500000

               B      A          7          38.000000

            C  B      A          2          49.000000

            C  B                 6          52.500000
```

```
       D     C                   5          65.000000

    E  D                         4          79.000000

    E                            3          88.000000

=======================================================
Significance level: 0.05
       Grouping              Treatment      Mean

                     A           1          30.000000

                     A           8          30.500000

                B    A           7          38.000000

             C  B                2          49.000000

          D  C                   6          52.500000

          D                      5          65.000000

    E                            4          79.000000

    E                            3          88.000000

=======================================================

++++++++++++++++++++++++++++++++++++++++++++
 End of analysis
++++++++++++++++++++++++++++++++++++++++++++
```

Concluding remarks

Stringent test procedures were followed throughout the development. The original sample data from [7] were repeatedly used to verify the C program during the coding process. After the C program was completed, a series of testing have been conducted. Sample data set were fed through both the BASIC program and the C program, identical results were always obtained indicating the validity of the C program. Furthermore, the C program has been successfully applied in a recent study comparing IPE (Individual Protection Ensemble) and COLPRO (Collective Protection). Results once again justified the usefulness of this analyzing tool.

Besides a structured usability study, future plans of this project also include the further development of the Friedman test and integrate it with other commercial available software packages.

To sum up, since the Friedman test is a distribution free test, there are no strong assumptions associated with the source data. It has been proved to be a powerful method for analyzing ranking data. The newly revamped C program resolved major usability problems existed in the previous BASIC program and enhanced its capabilities. This C program is expected to benefit more researchers in wide range of future studies.

References

1. Siegel, S and Castellan Jr., N. J. (1988). Nonparametric Statistics for the Behavioral Sciences. 2nd Edition. McGraw-Hill, New York.

2. Altman, D. G. (1991). Practical Statistics for Medical Research. Chapman and Hall, London.

3. Gibbons, J. D. (1993). Nonparametric Statistics: An Introduction. Newbury Park, CA: Sage.

4. SPSS version 13. SPSS, Inc.

5. SAS version 9. SAS Institute, Inc.

6. STATISTICA version 7, StatSoft, Inc.

7. Theodorsson-Norheim, E. (1987). Friedman and Quade Tests: Basic computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples. *Comput. Biol. Med.* Vol.17, 85-99.

8. Ho, G. W., and Dyck, W. (2003). Assessment of improved rain suits. Defence R&D Canada technical report, TR2003-116.

9. Ho, G. W., and Dyck, W. (2004). A validation study of the converged design CF rainsuit. Defence R&D Canada technical report, TR-2004-xxxxx.

10. Drolet, E., Ho, G. W. (2004). Skis, Bindings, and Poles. Defence R&D Canada technical report, TR-2004-xxxxx.

Annex

C Source code of Friedman test

```
/*************************************************************************
 * Friedman Quade Tests (modified from the original BASIC program)
 *
 * Reference: Theodorsson-Norheim, Elvar. (1987). Friedman and Quade Tests:
 *            Basic computer program to perform nonparametric two-way analysis
 *            of variance and multiple comparisons on ranks of several related
 *            samples. Comput. Biol. Med. Vol. 17, No. 2, pp.85-99.
 *            Printed in Great Britain.
 *
 * Renaming of some of the variables:
 *             BASIC          C
 *
 *             D[I, J]        raw_data[][]
 *             D1[I, J]       ranked_data[i][j]
 *             N[I%, 1]       sampled-range[i]
 *             N[I%, 2]       rank_of_sample_range[i]
 *             R[J%]          sum_of_ranks[j]
 *             D2             sij[][]
 *             M%             row
 *             N%             column
 *             B[I%]          B[]
 *
 * Compiled on Redhat 8.0: gcc friedman.c -lm -o friedman.exe
 *
 * Usage: friedman.exe data_file_name
 *
 * NOTE: 1. The raw data file should consisted of COMMA delimited data set.
 *       2. User shall ensure there is no missing data in the original file.
 *
 * Questions, contact: wenbi.wang@drdc-rddc.gc.ca
 *
 * Last Modifed: July 16, 2004
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

// define the maximum raw data matrix to be 1000x1000.
#define ROW 15
#define COL 15

float calculate_F(int, int, float);
float calculate_T(int, int, float);

int main(int argc, char** argv)
{
        FILE* raw_data_file;
        FILE* output_file;

        int row = 0;
        int column = 0;
        int i, j, k;
        float t3, t4, t5;
```

```
        float raw_data[ROW][COL];
        float ranked_data[ROW][COL];
        float sampled_range[ROW];
        float rank_of_sample_range[ROW];
        float sum_of_ranks[COL];
        float sij[ROW][COL];
        float a1 = 0;
        float b1 = 0;
        float B[COL];
        float t1, t9;
        double p;
        double e;
        float t6;
        float a2 = 0;
        float b2 = 0;
        float t2 = 0;
        float sorted_sum_of_ranks[COL];
        int sorted_column[COL];
        int counter;
        int current_group_no;
        int group_flag[COL][COL];     // assume each treatment can belong to maximun
'column' groups
        int stop_at = 1;              //  record  the  treatment  number  the  algorithm
previous stopped
                                      // at,  it  is  an  indication  of  the  new  group,
instead of a
                                      // subgroup of a previous one.
        int significance[ROW][COL];   //  keep  track  of  the  significance  between  any
paired
                                      // conditions

        // open up two files
        if(argc != 2) {
                printf("\nUsage:friedman_win.exe data_file\n\n");
                exit(1);
        }

        raw_data_file = fopen(argv[1], "r");
        if(raw_data_file == NULL) {
                printf("\nFailed to open raw data file!\n\n");
                exit(1);
        }

        output_file = fopen("output.txt", "a");
        if(raw_data_file == NULL) {
                printf("\nFailed to open output file!\n\n");
                exit(1);
        }

        fprintf(output_file,
"================================================================================
================================\n");
        fprintf(output_file, "Source data file for Friedman analysis: %s\n", argv[1]);
        fprintf(output_file,
"================================================================================
================================\n\n");

        // get the size of the raw data set
        while(!feof(raw_data_file)) {
                char ch;
                ch = fgetc(raw_data_file);
                if(ch == ',') {
                        if(row==0) column++; // determine the number of columns from the
first row
                }
                else if(ch == '\n') {
                        row++;         // count the current row number (start from 0)
```

```
		}
	}

	column++; // the last column wasn't added in the above routine

	fclose(raw_data_file);

	if(row >= ROW || column >= COL) {
		printf("Error:  Raw  data  exceed  program  processing  limit.  Contact
Wenbi.Wang@drdc-rddc.gc.ca\n");
		exit(1);
	}

	// print out the row, column numbers
	fprintf(output_file, "The size of the raw data matrix is: row = %d; column =
%d. \n\n", row, column);

	// read the raw data into the array


	raw_data_file = fopen(argv[1], "r");
	if(raw_data_file == NULL) {
		fprintf(output_file, "Failed to open raw data file!!\n");
		exit(1);
	}

	for(i=0; i<row; i++) {
		for(j=0; j<column; j++) {
			float data;
			fscanf(raw_data_file, "%f,", &data);
			raw_data[i][j] = data;
		}
	}

	fprintf(output_file, " A list of your raw data:\n\n");
	for(i=0; i<row; i++) {
		for(j=0; j<column; j++)
			fprintf(output_file, " %8.3f ", raw_data[i][j]);
		fprintf(output_file, "\n");
	}


	// rank the data and print them out

	for(i=0; i<row; i++) {
		float small, large, tmp;

		small = raw_data[i][0];
		large = raw_data[i][0];

		for(j=0; j<column; j++) {
			if(small > raw_data[i][j]) small = raw_data[i][j];
			if(large < raw_data[i][j]) large = raw_data[i][j];
			tmp = 0.5;
			for(k=0; k<column; k++) {
				if(raw_data[i][k] < raw_data[i][j]) tmp++;
				else if(raw_data[i][k] == raw_data[i][j]) tmp+=0.5;
			}
			ranked_data[i][j] = tmp;
		}
		sampled_range[i] = large - small;
	}

	fprintf(output_file, "\nA list of your ranked data:\n\n");

	////////////////// rank overservation ranges and store
```

```
        for(i=0; i < row; i++) {
                float tmp = 0.5;

                for(j=0; j<row; j++) {
                        if(sampled_range[j] < sampled_range[i]) tmp++;
                        if(sampled_range[j] == sampled_range[i]) tmp+=0.5;
                }
                rank_of_sample_range[i] = tmp;
        }

        fprintf(output_file, "\nRank of sample ranges:\n\n");
        for(i=0; i<row; i++) {
                for(j=0; j<column; j++)
                        fprintf(output_file, " %8.3f ", ranked_data[i][j]);
                fprintf(output_file,  "\tsample  range  =  %8.3f,\t  rank  of  sample
range:%3.0f\n", sampled_range[i], rank_of_sample_range[i]);
        }

        /////////////// sum of ranks for each treatment


        for(j=0; j<column; j++) {
                sum_of_ranks[j]=0;
                for(i=0; i<row; i++)
                        sum_of_ranks[j] = sum_of_ranks[j] + ranked_data[i][j];
        }


        fprintf(output_file, "\nSum of ranks for each treatment:\n\n");
        for(j=0; j<column; j++) {
                        fprintf(output_file, " %8.3f ", sum_of_ranks[j]);
        }
        fprintf(output_file, "\n");


        /////////////// Calculate Sij


        for(i=0; i<row; i++)
                for(j=0; j<column; j++)
                        sij[i][j]=rank_of_sample_range[i]*(ranked_data[i][j]-(column/2 +
0.5));

        fprintf(output_file, "\nList of Sij:\n\n");

        for(i=0; i<row; i++) {
                for(j=0; j<column; j++)
                        fprintf(output_file, " %8.3f ", sij[i][j]);
                fprintf(output_file, "\n");
        }

        /////////////// Calculate a1 for the quade test



        for(i=0; i<row; i++)
                for(j=0; j<column; j++)
                        a1 = a1 + sij[i][j] * sij[i][j];

        fprintf(output_file, "\n\n*******************************\n");
        fprintf(output_file, "Resules from the Quade Test\n");
        fprintf(output_file, "The Quade Test A1 = %8.3f\n", a1);

        /////////////// Calculate b1 for the quade test
```

```
        for(i=0; i<column; i++) {
                B[i] = 0;
                for(j=0; j<row; j++) {
                        B[i]+= sij[j][i];
                }
                b1 += B[i]*B[i];
        }
        b1 /= row;

        fprintf(output_file, "The Quade Test B1 = %8.3f\n", b1);

        /////////////// Calculate t1 for the quade test



        t1 = ((row-1)*b1)/(a1-b1);

        fprintf(output_file, "The Quade Test T1 = %8.3f", t1);

        t9 = t1;

        //////// calculate T distribution quantiles (by calling calculate_T())
        t3 = calculate_T(row, column, 0.05);
        t4 = calculate_T(row, column, 0.01);
        t5 = calculate_T(row, column, 0.001);

        ////////////// call the calculate_F() function


        p = calculate_F(row, column, t9);
        fprintf(output_file, "        p = %e\n", p);

        ///////////////// Do not perform multiple comparisons if
        ///////////////// overall test is not significant
        if(p<=0.05) {
                t6 = sqrt((2*row*(a1 - b1))/((row - 1)*(column - 1)));
                fprintf(output_file, "\nDifference between sums for Quade Test\n\n");
                for(i=0; i<column; i++) {
                        fprintf(output_file, "Observation %d\n", i+1);
                        for(j=0; j<column; j++) {
                                e = B[i]-B[j];
                                if(e<0) e*=-1;
                                fprintf(output_file, "      - observation %d = %8.3f",
j+1, e);
                                if(e>t6*t5) fprintf(output_file, "   p<0.001\n");
                                else if(e>t6*t4) fprintf(output_file, "   p<0.01\n");
                                else if(e>t6*t3) fprintf(output_file, "   p<0.05\n");
                                else fprintf(output_file, "   NS\n");
                        }
                }
        }



        ///////////////// calcualte a2 for the friedman test


        for(i=0; i<row; i++)
                for(j=0; j<column; j++)
                        a2 = a2 + ranked_data[i][j]*ranked_data[i][j];

        fprintf(output_file, "\n\n\n*******************************\n");
        fprintf(output_file, "Results from the Friedman test\n");

        fprintf(output_file, "The Friedman test A2 = %8.3f\n", a2);

        ////////////// Calculate b2 for the friedman test
```

```
        for(j=0; j<column; j++)
                b2 = b2 + sum_of_ranks[j]*sum_of_ranks[j];

        b2 /= row;

        fprintf(output_file, "The Friedman test B2 = %8.3f\n", b2);

        ////////////////////// calculate T2 for the Friedman test


        t2 = ((row-1)*(b2-(row*column*((column+1)*(column+1))/4)))/(a2-b2);

        fprintf(output_file, "the Friedman test T2 = %8.3f", t2);

        t9 = t2;

        ////////////////// call the calculate_F() function
        p = calculate_F(row, column, t9);
        fprintf(output_file, "        p = %e\n", p);

        ////////////////////////// Do not perform multiple comparisons if overall test
is not significant
        e = 0;

                                        // p<0.001   3
                                        // p<0.05    2
                                        // p<0.01    1
                                        // NS        0

        if (p<=0.05) {
                t6 = sqrt((2*row*(a2 - b2))/((row-1)*(column-1)));
                fprintf(output_file,   "\nDifference   between   sums   for   Friedman
test:\n\n");
                for(i=0; i<column; i++) {
                        fprintf(output_file, "Observation %d\n", i+1);
                        for(j=0; j<column; j++) {
                                e=sum_of_ranks[i]-sum_of_ranks[j];
                                if(e<0) e*=-1;
                                fprintf(output_file, "    -observation %d = %8.3f", j+1,
e);
                                if(e>t6*t5) {
                                        fprintf(output_file, "   p<0.001\n");
                                        significance[i][j] = 3;
                                }
                                else if(e>t6*t4) {
                                        fprintf(output_file, "   p<0.01\n");
                                        significance[i][j] = 2;
                                }
                                else if(e>t6*t3) {
                                        fprintf(output_file, "   p<0.05\n");
                                        significance[i][j] = 1;
                                }
                                else {
                                        fprintf(output_file, "   NS\n");
                                        significance[i][j] = 0;
                                }
                        }
                }
        }

        ////////////////////////////////////////
        ////////////////// print out significance table
        ////////////////////////////////////////

        /* for(i=0; i<column; i++) {
```

```
		for(j=0; j<column; j++) {
			printf("\t%f", significance[i][j]);
		}
		printf("\n");
	}*/

	////////////////////// Grouping

	fprintf(output_file, "\n\n\nMeans with the same same grouping letters are NOT
significantly different!\n");

	/// sorting the sum_of_ranks[]


	for(i=0; i<column; i++) {
		counter = 0;
		for(j=0; j<column; j++) {
			if(sum_of_ranks[i] > sum_of_ranks[j]) counter++;
			if((sum_of_ranks[i] == sum_of_ranks[j]) && (i>j)) counter++;
		}
	//	printf("counter=%d\n", counter);
		sorted_sum_of_ranks[counter] = sum_of_ranks[i];
		sorted_column[counter] = i;
	//	printf("%f,		%d\n",		sorted_sum_of_ranks[counter],
sorted_column[counter]);
	}
	// at this point, sorted_sum_of_ranks[] should be ready and sorted_column[]
	//  has  the  associated  origainal  column  number  stored  wrt  to  the
sorted_sum_of_ranks[]


////////// code here copied three times to calculate different significance level
(p<0.001)
	fprintf(output_file,
"=====================================================\n");
	fprintf(output_file, "Significance level: 0.001\n");
	fprintf(output_file, "\tGrouping\t\tTreatment\tMean\n\n");

	for(i=0; i<column; i++)
		for(j=0; j<column; j++)
			group_flag[i][j] = 0;

	current_group_no = 1; // initial group starts from 1

	for(i=0; i<column-1; i++) {
		if((significance[sorted_column[i]][sorted_column[stop_at]]		<
3)&&(stop_at != -1)) { //test if the new tested group is not a subgroup of previous
ones
			for(k=i; k<column; k++)
				if(group_flag[sorted_column[i]][k] == 0) {
					group_flag[sorted_column[i]][k]		=
current_group_no;
					break;
				}

			for(j=i+1; j<column; j++) {
				if(significance[sorted_column[i]][sorted_column[j]] < 3)
{

					for(k=i; k<column; k++)
						if(group_flag[sorted_column[i]][k]  ==  0)
{
							group_flag[sorted_column[i]][k]  =
current_group_no;
							break;
						}
```

```
                                        if(j==column - 1) stop_at = -1; // the end of
treatment has been reached, no need for further grouping

                                }
                                else {
                                        stop_at = j;
                                        break;
                                }
                        }
                        current_group_no++;
                }
        }

/*      for(i=0; i<column; i++) {
                for(j=0; j<column; j++) {
                        printf("%d\t", group_flag[sorted_column[i]][j]);
                }
                printf("\n");
        }

        printf("\n");
*/
        for(i=0; i<column; i++) {
                for(j=column-1; j>=0; j--) {
                        if(group_flag[sorted_column[j]][i] == 0)
                                fprintf(output_file,         "                          ",
group_flag[sorted_column[j]][i]);
                        else                    fprintf(output_file,     "     %c     ",
group_flag[sorted_column[j]][i]+64); // print out letters instead of numbers
                }
                fprintf(output_file,  "                    %d                    %f\n\n",
sorted_column[i]+1, sorted_sum_of_ranks[i]);
        }
///////////////// end of significance level (p<0.001)

////////////  code  here  copied  three  times  to  calculate  different  significance  level
(p<0.01)
        fprintf(output_file,
"=====================================================\n");
        fprintf(output_file, "Significance level: 0.01\n");
        fprintf(output_file, "\tGrouping\t\tTreatment\tMean\n\n");

        for(i=0; i<column; i++)
               for(j=0; j<column; j++)
                        group_flag[i][j] = 0;

        current_group_no = 1; // initial group starts from 1
        stop_at = 1;

        for(i=0; i<column-1; i++) {
                if((significance[sorted_column[i]][sorted_column[stop_at]]             <
2)&&(stop_at!=-1)) { //test if the new tested group is not a subgroup of previous ones
                        for(k=i; k<column; k++)
                                if(group_flag[sorted_column[i]][k] == 0) {
                                        group_flag[sorted_column[i]][k]                =
current_group_no;
                                        break;
                                }

                        for(j=i+1; j<column; j++) {
                                if(significance[sorted_column[i]][sorted_column[j]] < 2)
{

                                        for(k=i; k<column; k++)
                                                if(group_flag[sorted_column[i]][k]  ==  0)
{
```

```
                                                group_flag[sorted_column[i]][k]  =
current_group_no;
                                                break;
                                        }
                                if(j==column - 1) stop_at = -1; // the end of
treatment has been reached, no need for further grouping

                        }
                        else {
                                stop_at = j;
                                break;
                        }
                }
                current_group_no++;
        }
}

/*      for(i=0; i<column; i++) {
                for(j=0; j<column; j++) {
                        printf("%d\t", group_flag[sorted_column[i]][j]);
                }
                printf("\n");
        }

        printf("\n");
*/
        for(i=0; i<column; i++) {
                for(j=column-1; j>=0; j--) {
                        if(group_flag[sorted_column[j]][i] == 0)
                                fprintf(output_file,         "                          ",
group_flag[sorted_column[j]][i]);
                        else                    fprintf(output_file,     "     %c     ",
group_flag[sorted_column[j]][i]+64);
                }
                fprintf(output_file,  "                    %d                    %f\n\n",
sorted_column[i]+1, sorted_sum_of_ranks[i]);
        }
///////////////// end of significance level (p<0.01)

////////////  code here copied three times to calculate different significance level
(p<0.05)
        fprintf(output_file,
"====================================================\n");
        fprintf(output_file, "Significance level: 0.05\n");
        fprintf(output_file, "\tGrouping\t\tTreatment\tMean\n\n");

        for(i=0; i<column; i++)
               for(j=0; j<column; j++)
                        group_flag[i][j] = 0;

        current_group_no = 1; // initial group starts from 1
        stop_at = 1;

        for(i=0; i<column-1; i++) {
                if((significance[sorted_column[i]][sorted_column[stop_at]]             <
1)&&(stop_at!=-1)) { //test if the new tested group is not a subgroup of previous ones
                        for(k=i; k<column; k++)
                                if(group_flag[sorted_column[i]][k] == 0) {
                                        group_flag[sorted_column[i]][k]                =
current_group_no;
                                        break;
                                }

                        for(j=i+1; j<column; j++) {
                                if(significance[sorted_column[i]][sorted_column[j]] < 1)
{
```

```
                                        for(k=i; k<column; k++)
                                                if(group_flag[sorted_column[i]][k]  ==  0)
{
                                                        group_flag[sorted_column[i]][k]  =
current_group_no;
                                                        break;
                                                }
                                        if(j==column  -  1)  stop_at  =  -1;  //  the  end  of
treatment has been reached, no need for further grouping

                                }
                                else {
                                        stop_at = j;
                                        break;
                                }
                        }
                        current_group_no++;
                }
        }

/*      for(i=0; i<column; i++) {
                for(j=0; j<column; j++) {
                        printf("%d\t", group_flag[sorted_column[i]][j]);
                }
                printf("\n");
        }

        printf("\n");
*/
        for(i=0; i<column; i++) {
                for(j=column-1; j>=0; j--) {
                        if(group_flag[sorted_column[j]][i] == 0)
                                fprintf(output_file,         "                          ",
group_flag[sorted_column[j]][i]);
                        else                    fprintf(output_file,     "     %c     ",
group_flag[sorted_column[j]][i]+64);
                }
                fprintf(output_file,  "                    %d                    %f\n\n",
sorted_column[i]+1, sorted_sum_of_ranks[i]);
        }
        fprintf(output_file,
"=====================================================\n");
///////////////// end of significance level (p<0.05)

        fprintf(output_file, "\n\n+++++++++++++++++++++++++++++++++++++++++++++++\n");
        fprintf(output_file, " End of anlaysis\n");
        fprintf(output_file, "+++++++++++++++++++++++++++++++++++++++++++++++\n\n");

        fclose(raw_data_file);
        fclose(output_file);


        return 0;
}

float calculate_F(int row, int col, float t9) {
        double c1, c2, c3, c4, c5, c6, c7, c8, c9;
        double p = 0;

        c3 = 0;
        c1 = col - 1;
        c2 = (col-1)*(row-1);

// truncl is a GCC building function, use floorl instead in generating windows code.
//      if(c1 == (2*truncl(c1/2 + 0.1))) {
        if(c1 == (2*(floorl(c1/2 + 0.1)))) {
                c4 = 1/(1+c2/(t9*c1));
```

```
                c5 = c1 + 1;
                c6 = c2 - 2;
                c7 = 0;
                c8 = 1;
                c9 = 2;
                do {
                        c7 = c7 + c8;
                        c8 = c8 * c4 * (c9+c6)/c9;
                        c9 = c9 + 2;
                } while (c9<c5);
                if(c3==0) {
                        p = 100*c7*(pow(sqrt(1-c4), c2));
                } else {
                        p = 100*c7*(pow(sqrt(1-c4), c1));
                }
//      } else if(c2 == 2*truncl(c2/2+0.1)) {
        } else if(c2 == 2*(floorl(c2/2+0.1))) {
                c3 = 1;
                c4 = 1/(1+t9*c1/c2);
                c5 = c2 + 1;
                c6 = c1 - 2;
                c7 = 0;
                c8 = 1;
                c9 = 2;
                do {
                        c7 = c7 + c8;
                        c8 = c8 * c4 * (c9+c6)/c9;
                        c9 = c9 + 2;
                } while (c9<c5);
                if(c3==0) {
                        p = 100*c7*(pow(sqrt(1-c4), c2));
                } else {
                        p = 100*c7*(pow(sqrt(1-c4), c1));
                }
        } else {
                c4 = 1/(1+t9*c1/c2);
                c7 = 0;
                c8 = 1;
                c9 = 2;
                c5 = c2;
                while (c9 < c5) {
                        c7 = c7 + c8;
                        c8 = c8*c4*c9/(c9+1);
                        c9 = c9+2;
                }
                c8 = c8*c2;
                c9 = 3;
                c5 = c1 + 1;
                c6 = c2 - 2;
                while (c9 < c5) {
                        c7 = c7 - c8;
                        c8 = c8*(1-c4)*(c9+c6)/c9;
                        c9 = c9+2;
                }
                t9 = atan(sqrt(t9*c1/c2));
                p = 100*(1-2*(t9+c7*sqrt((1-c4)*c4))/3.14159);
        }

        if(p<=50) {
                p = p/100;
                if (p<0) p*=-1;
        } else {
                p = 100 - p;
                p = p/100;
                if (p<0) p*=-1;
        }
```

```
	return p;
}

/////////////// calculate T distribution quantiles
float calculate_T(int row, int col, float q) {
	float q1 = q;
	float q2 = (col - 1)*(row - 1);

	float q3, q4, q5, q6, q7, q8, q9, y, y1, y2, y3, y4, t;

	if(q2 > 2) {
		q4 = 1/(q2-0.5);
		q5 = 48/(q4*q4);
		q6 = ((20700*q4/q5-98)*q4-16)+96.36;
		q7 = ((94.5/(q5+q6)-3)/q5+1)*sqrt(q4*1.5708)*q2;
		q8 = q7*q1;
		y  = pow(q8, (2/q2));

		if(y > 0.05+q4) {
			q9 = sqrt(-2*log(q1));
			q8 = 2.51552+q9*(0.802853+0.010328*q9);
			q8 = q9 - q8/(1+q9*(1.43279+q9*(0.189269+0.001308*q9)));
			y  = (2*q8)*(2*q8);
			if(q2<5)
				q6 = q6 + 0.3*(q2-4.5)*(q8+0.6);
			else
				q6 = (((0.05*q7*q8-5)*q8-7)*q8-2)*q8+q5+q6;
			y  = (((((0.4*y+6.3)*y+36)*y+94.5)/q6-y-3)/q5+1)*q8;
			y  = q4*y*y;
			if(y >0.002) y = exp(y) - 1;
			if(y<=0.002) y = 0.5*y*y+y;
			t = sqrt(q2*y);
		} else {
			y1 = (q2+2)*3;
			y2 = q2+4;
			y3 = q2+1;
			y4 = q2+6;
			y           =     ((1/((y4/(q2*y)-0.089*q7-0.822)*y1)+0.5/y2)*y-
1)*y3/(q2+2)+1;
			t  = sqrt(q2*y);
		}
	} else {
		if(q2 == 1) t = cos(q1*1.5708)/sin(q1*1.5708);
		if(q2 == 2) t = sqrt(2/(q1*(2-q1))-2);
	}

	return t;

}
```

DOCUMENT CONTROL DATA
(Security classification of the title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document, Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Publishing: DRDC Toronto
Performing: DRDC Toronto
Monitoring:
Contracting:

2. SECURITY CLASSIFICATION
(Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification is indicated by the appropriate abbreviation (S, C, R, or U) in parenthesis at the end of the title)

Friedman Test: a C Program for Nonparametric Two−way Analysis of Variance and Multiple Comparisons on Ranks (U)

4. AUTHORS (First name, middle initial and last name. If military, show rank, e.g. Maj. John E. Doe.)

Wenbi Wang, Walter R. Dyck, Ghee W. Ho

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
December 2004	35	

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The names of the department project office or laboratory sponsoring the research and development − include address.)

Sponsoring:
Tasking:

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document) DRDC Toronto TM 2004−214	10b. OTHER DOCUMENT NO(s). (Any other numbers under which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILIY (Any limitations on the dissemination of the document, other than those imposed by security classification.)

Unlimited distribution

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11), However, when further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited announcement

DOCUMENT CONTROL DATA
(Security classification of the title, body of abstract and indexing annotation must be entered when the overall document is classified)

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

(U) The Friedman test is also known as a two−way analysis on ranks. It is a nonparametric analysis method frequently used to analyze field and laboratory trial data in Human System Engineering group at DRDC Toronto. A BASIC program was previously developed to automate the data processing. However, due to the constraints of the programming language, the BASIC program has several intrinsic usability problems. This project was initiated with a goal to resolve these problems. Rewritten in C language, the new Friedman test program has improved its input/output capabilities, will perform post−hoc pair−wise treatment comparisons, and is able to run on multiple Operating Systems. This new program has successfully passed tests in several recent DRDC studies. It has been proved to be a useful analyzing tool and will benefit researchers in a wider range of future studies.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

(U) Friedman test; Non−parametric statistics; Multiple comparisons techniques

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca