Journal of Modern Applied Statistical Methods

Volume 1 | Issue 2 Article 58

11-1-2002

JMASM4: Critical Values For Four Nonparametric And/Or Distribution-Free Tests Of Location For Two Independent Samples

Bruce R. Fay
Wayne County Regional Educational Service Agency, Michigan

Journal of Modern Applied Statistical Methods
Fall 2002, Vol. 1, No 2, 489-517

1538 – 9472/02/$30.00

JMASM Algorithms and Code
JMASM4: Critical Values For Four Nonparametric And/Or Distribution-Free Tests Of Location For Two Independent Samples

Bruce R. Fay
Assessment & Evaluation
Wayne County *R*egional *E*ducational *S*ervice *A*gency

Researchers engaged in computer-intensive studies may need exact critical values, especially for sample sizes and alpha levels not normally found in published tables, as well as the ability to control 'best-fit' criteria. They may also benefit from the ability to directly generate these values rather than having to create lookup tables. Fortran 90 programs generate 'best-conservative' (bc) and 'best-fit' (bf) critical values with associated probabilities for the Kolmogorov-Smirnov test of general differences (bc), Rosenbaum's test of location (bc), Tukey's quick test (bc and bf)) and the Wilcoxon rank-sum test (bc).

Key words: Kolmogorov-Smirnov test, Rosenbaum test, Tukey quick test; Wilcoxon rank-sum test.

Introduction

Researchers, especially those engaged in Monte Carlo studies, may have a need for exact critical values over a wider range of sample sizes and/or alpha levels than are generally available from published tables. They may also benefit from the ability to generate the values directly, as opposed to creating lookup tables, and to control best-fit criteria. Fortran 90 programs that generate critical values for four nonparametric/distribution-free tests of location for two independent samples are presented. Included are the Kolmogorov-Smirnov test of general differences, Rosenbaum's test of location, Tukey's quick test and the Wilcoxon rank-sum test. The programs for Tukey's test also generate 'best-fit' critical values and associated probabilities. The best-fit method could be adapted to the other programs.

Bruce R. Fay is an Assessment Consultant. He works with K-12 public schools in school accountability, accreditation, and assessment of student learning. Contact him at 30580 Springland St., Farmington Hills, MI 48334 or by e-mail at bfay@twmi.rr.com.

Tukey Quick Test

Tukey (1959) described a method for generating critical values for his *Two-Sample Test to Duckworth's Specifications*, now commonly known as Tukey's Quick Test. The test is both quick and compact, which makes it portable. The "rule of thumb" critical values, however, are not consistently 'best-conservative' or 'best-fit' to specific criteria.

Test Description

Tukey's (1959) test is quick in the sense that the method is easily remembered and the statistic, based on the combined length of extreme runs, easily calculated. The two samples are combined and ordered. For a two-sided test, if the overall maximum and minimum come from different groups, the statistic is the number of observations from the group with the global maximum that are greater than the greatest observation from the group with the global minimum plus the number of observations from the group with the global minimum that are less than the least observation from the group with the global maximum. If the global maximum and minimum are from the same group the statistic is generally taken to be zero. Tukey (1959) suggested dealing with ties (consequential, between-group) by counting each tied observation as ½ rather than 1. The one-sided (directional) test statistic is calculated just like the two-sided statistic with the additional requirement that the overall maximum observation is from the group that is expected to have the higher median under the alternative hypothesis (assuming a pure shift model). If not, the statistic is taken to be zero.

The test is compact in the sense that the critical values do not vary much with sample size, especially if the sample sizes are not too different. As such, they can also be easily committed to memory. For two-sided tests at nominal alpha levels of .10, .05, .02 and .01 (or one-sided tests at .05, .025, .01 and .005) the best-conservative critical values are 6, 7, 9 and 10 respectively with equal sample sizes from 9 to 24 per group. Tukey (1959) suggested that these critical values be used for all sample sizes as long as they were not too different. He noted, however, that under these conditions the test was not strictly conservative in the classical sense. He also gave relatively simple corrections to apply when the sample sizes were different, although not by too much. These corrections, however, still do not guarantee that the test will be strictly conservative, and add a level of complexity to the test that reduces both its quickness and compactness.

The best-fitting critical values for nominal alpha levels (1-sided) of .05, .025, .01, .005 (with a +10% tolerance) are 6, 7, 8 and 9 for equal samples sizes from 5 to 9 and 6, 7, 9, 10 for equal sample sizes from 11 to 30. Using 6, 7, 9, 10 as the critical values for all equal sample sizes is conservative for samples sizes less than 11 at .02 and .01 alpha levels (2-sided) but may be liberal up to +10% for other sample sizes and nominal alphas.

Quickness and compactness combine to make Tukey's (1959) test portable in the sense that everything needed to apply the test can be carried around in one's memory and the calculations can be performed mentally, or with pencil and paper. This simplicity is gained at the expense of some statistical power, but the practical power may be high. Tukey (1959) referenced a definition of practical power from Churchill Eisenhart (without formal citation) as "the product of the mathematical power by the probability that the procedure will be used" and noted that the practical power of a test might prove to be quite high, in spite of lower statistical power, if it became widely used.

Because of its portability and potentially high practical power, Tukey (1959) referred to this test as a "pocket test" and proposed that it filled a particular niche, i.e., "as a footrule", "on the floor", or "in the field" to "indicate the weight of the evidence roughly." He recommended that more sensitive tests be used "if a delicate and critical decision is to be made."

Methodology for Generating Critical Values and Associated Probabilities

Tukey (1959) described in detail a method for generating strictly conservative, exact critical values. That method is implemented in the program modules presented here, along with a variation that produces best-fitting critical values to a specified tolerance level above nominal alpha.

Tukey's (1959) method involves building a table, A, that contains "a certain summation of binomial coefficients." Differences of pairs of entries from A, based on the sample sizes j and k and a parameter h, are compared to ${}_n\mathrm{C}_j$, the number of combinations of n things taken j at a time, where $n = j + k$, $j \geq 1$, $k \geq 1$, and $j \leq k$. The differences $A(k - h, j) - A(k, j - h)$ are formed starting with $h = 1$ and counting up until the difference is less than (nominal alpha)x(${}_n\mathrm{C}_j$). The first such value of h, if one exists, is the best-conservative critical value for that pair of sample sizes and nominal alpha level. Additional details of the method are given in the comments that accompany the programs. Based on the use of integer*8 and real*8 variables, critical values and associated probabilities are generated for all combinations of sample sizes from (1, 1) to (30, 30) in increments of 1 for each sample. Tukey (1959) also presented asymptotic methods that may be appropriate for larger sample sizes.

The module that generates the critical values and associated probabilities contains two versions of the method and a subroutine for calculating combinations. The first version of the method generates strictly conservative critical values for one-sided tests at .05, .025, .01 and .005 nominal alpha levels. The second version generates 'best-fit' critical values for one-sided tests at the same nominal alpha levels. The 'best-fit' version allows critical values greater than nominal alpha so long as they do not exceed nominal alpha by more than 10% and are closer to nominal alpha than the nearest value that is less than nominal alpha. The +10% tolerance is based on a definition of robustness due to Bradley (1978).

Rosenbaum's Test of Location

Rosenbaum (1953, 1954) described tests for dispersion and location based on Wilks (1942) and gave tables of critical values. Rosenbaum (1965) revisited these tests, comparing them to other tests that had arisen in the intervening decade. Neave & Worthington (1988) described the location form of the test as particularly well suited to situations in which spread is expected to increase with an increase in the median and gave a method for generating critical values. Their method is the basis for the programs presented here. Rosenbaum's (1954) test is quick and relatively compact, which makes it somewhat portable.

Test Description

The test is quick in the sense that the method is easily remembered and the statistic, based on the length of an extreme run, easily calculated. The two samples are combined and ordered. For a two-sided test, the statistic is taken as the number of observations from the group with the overall maximum that exceeds the maximum value of the other group. One way to deal with consequential (between-group) ties is to count each observation as ½ rather than 1. Another method is to average the values of the statistic arrived at by resolving the ties in all possible ways. The later technique, however, causes the test to lose some of its portability, at least for larger sample sizes. The one-sided (directional) test statistic is calculated just like the two-sided statistic with the additional requirement that the overall maximum observation is from the group that is expected to have the higher median under the alternative hypothesis (assuming a pure shift model). If not, the statistic is taken to be zero.

The test is compact in the sense that the critical values do not vary much with sample size, especially if the sample sizes are not too different. As such, they can also be easily committed to memory. For two-sided tests at nominal alpha levels of .10, .05, .02 and .01 (or one-sided tests at .05, .025, .01 and .005) the best-conservative critical values are 5, 6, 7 and 8 respectively for equal sample sizes from 27 to 50 per group. Critical values of 5, 6, 7, and 8 can be used for equal sample sizes from 20 to 50, and critical values of 4, 5, 6 and 7 for equal sample sizes from 5 to 19, if one is willing to accept results that are not strictly conservative in all cases, and somewhat overly conservative in others. Under these conditions the test can be considered compact. Quickness and compactness combine to make the test portable as previously described.

Methodology for Generating Critical Values and Associated Probabilities

Neave & Worthington (1988) described a method for generating strictly conservative, exact critical values. Their method is implemented in the program modules presented here to calculate the critical values for one-sided tests at .05, .025, .01 and .005 nominal alpha levels.

Neave & Worthington (1988) calculated the probability of a run of h values from a sample of size m out of a combined sample of size $N = m + n$, where n is the size of the other group, using the formula:

$$\frac{m!(N-h)!}{N!(m-h)!} = \frac{m}{N} \times \frac{m-1}{N-1} \times \cdots \times \frac{m-h+1}{N-m+1}. \tag{1}$$

The value of h associated with the largest such probability that is less than or equal to nominal alpha is the critical value for a given m and n. Thus all critical values are best-conservative with pr(CV) ≤ nominal alpha. Additional details of the method are given in the comments that accompany the programs. Based on the use of integer*8 and real*8 variables, critical values and associated probabilities are generated for all combinations of sample sizes from (1, 1) to (50, 50) in increments of 1 for each sample.

Kolmogorov-Smirnov Test of General Differences

Kim and Jennrich (1970, 1973) cited Smirnov (1939) as introducing the criterion D_{mn} for the two-sample problem. As the name implies, the test is sensitive to general differences between two populations and is often used as a 2-sided test. Neave and Worthington (1988) pointed out, however, that the test functions quite well as a directional (1-sided) test, especially against a pure shift alternative. Kim and Jennrich (1970, 1973) provided a brief review of work on approximate and exact distributions of the statistic and resultant critical values under the null hypothesis leading up to their method and tables.

Test Description

The 2-sided test is conducted by constructing and then comparing the empirical cumulative distributions, $S_m(x)$ and $S_n(x)$, of two samples of size m and n ($m \le n$ without loss of generality) and then computing the criterion as $D_{mn} = \sup \mid S_m(x) - S_n(x) \mid$ over all x. The null hypothesis is that the two samples are drawn from identical (continuous) populations $F_m(x)$ and $F_n(x)$ (of any shape). The alternative hypothesis is that the samples were drawn from two populations that differ in some way. For a 1-sided test under a pure shift model, the criterion is taken to be D_{mn}^+ or D_{mn}^- , where $D_{mn}^+ = \max [\ S_m(x) - S_n(x)\ ] \ge 0$ and $D_{mn}^- = \min [\ S_m(x) - S_n(x)\ ] \le 0$. The choice depends on which sample is presumed to come from the population with the higher median under the alternative hypothesis. If the alternative hypothesis is that the samples came from populations with cumulative distributions such that $F_n(x) \ge F_m(x)$ then $S_n(x)$ will lie to the right of $S_m(x)$. Thus, $S_m(x)$ will rise faster than $S_n(x)$ and lie above it for any given value of x. This makes D_{mn}^+ the correct choice of criterion in this case.

Methodology for Generating Critical Values and Associated Probabilities

The Kim and Jennrich (1970, 1973) method of generating critical values for the Kolmogorov- Smirnov test is based on the work of Kim (1969) which, in turn, was an extension of the successive recursion relation of Massey (1951). Their method calculates:

$$P\left(D_{mn} \le \frac{c}{mn}\right) = U(m,n) \tag{2}$$

where

$$U(i,j) = \frac{i}{i+n} C(i,j)\left[U(i,j-1) + U(i-1,j)\right] \tag{3}$$

and

$$C(i,j) = \begin{cases} 1 \text{ if } \left|\frac{i}{m} - \frac{j}{n}\right| \le \frac{c}{mn} \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

subject to initial condition

$$U(i,j) = \binom{i+n}{i}^{-1} C(i,j),\ when\ i \bullet j = 0\,. \tag{5}$$

Kim and Jennrich (1970, 1973) provided a FORTRAN IV function subroutine *ASKCDF*(*M,N,D,U*) that returned the probability of D (= c/mn) for sample sizes m and n by calculating $U(m,n)$ as above. The U referenced in their function subroutine argument list, however, was merely a working storage vector of at least length N+1. In the Fortran 90 implementation of *ASKCDF* that follows, the working storage vector argument has been eliminated and replaced in the code with an allocatable array. A subroutine calculates $D = c/mn$ for $c = (1,mn,1)$ for each combination of $n = (1,50,1)$ and $m = (1,n,1)$ and calls *ASKCDF* for each value of D to obtain the probability and tests it against various nominal alpha levels.

Wilcoxon Rank-sum Test

Wilcoxon (1945) introduced the non-parametric/distribution-free test based on a sum of ranks that bears his name. Wilcoxon (1946, 1947) expanded on this work, followed by Mann and Whitney (1947), who described a test that turned out to be equivalent to the rank-sum test. The Wilcoxon-Mann-Whitney test is probably the best known of the nonparametric/distribution-free procedures. However, the early work of both Wilcoxon and Mann-Whitney provided only limited critical values. Additional work on both exact and approximate critical values and significance probabilities followed these seminal articles, e.g. Fix & Hodges (1955).

Jacobson (1963) provided a nice synopsis of critical value tables and work-to-date with an extensive bibliography. Wilcoxon and Wilcox (1964, revised 1968) provided a workable method for generating critical values and probability levels. This work subsequently appeared in Wilcoxon, Katti and Wilcox (1970, revised 1973) and forms the basis for the programs presented here.

Test Description

The Wilcoxon rank-sum version of the test is conducted by combining the observations from two samples. The combined samples are then ranked while keeping track of the original group membership. The ranks from one of the groups are then summed to form the statistic. Which group to sum for a 1-sided test depends on the critical value tables that are available (lower-tail, upper tail, or both) and on which group is expected to have the least (or greatest) ranks under the alternative hypothesis. For example, if lower tail critical values are available, and the alternative hypothesis is that sample *B* comes from a population that is greater than the population from which sample *A* was obtained, then sample *A* will tend to have the lower ranks, and the sum of those ranks would be taken as the statistic. For a two- sided test, one would form the sum of the ranks of both samples and test the resulting values against the critical value, taking the test to be significant if either comparison so indicated.

Methodology for Generating Critical Values and Associated Probabilities

Although critical values are readily available for the Wilcoxon rank-sum test and Mann-Whitney *U* test, the probability levels are not as accessible. The method of Wilcoxon, Katti and Wilcox (1970, 1973) proceeds along the following lines given samples *M* and *N* from two continuous populations, $F_m(x)$ and $G_n(x)$ of size m and n respectively, $m \le n$ without loss of generality. The minimum sum of ranks for sample *M* is $m(m+1)/2$. Thus the sum of ranks in general for sample *M* is:

$$\frac{m(m+1)}{2} + U \text{ where } U \in I,\ U \ge 0\,. \tag{6}$$

The number of ways, $f(U)$, of obtaining a specific rank sum U, is the coefficient of t^U in the expansion of the generating function, in powers of t, given by:

$$g(t) = \prod_{i=1}^{n} \frac{\left(1 - t^{m+i}\right)}{\left(1 - t^{i}\right)}\,. \tag{7}$$

The total number of ways of obtaining any rank sum in this situation is:

$$T = \binom{m+n}{n}\,. \tag{8}$$

Given $F_m(x) \equiv G_n(x)$, the probability of obtaining U is given by:

$$pr(U) = \frac{f(U)}{T}\,. \tag{9}$$

In turn, f(U) can be found from:

$$f(U) = \frac{1}{U}\sum_{i=0}^{U-1} f(i)\, z_{U-i-1} \quad \text{for } (U = 1,2,3,\ldots) \text{ and with } f(0) = 1 \tag{10}$$

In order to evaluate equation (10) it is necessary to find the values of z. Subroutine CV_WRSJ4_init in module CVWRSJmod includes the code for generating the values of z.

Source Code and Computing Platforms

All source code provided here is Fortran 90 free format. For each of the four tests there is a module that contains the critical value generation subroutines and functions and a main program that can be used with that module to generate printed tables of critical values and probabilities. The programs were developed on a 500 MHz AMD Athlon-based system using Compaq Visual Fortran 6.6 and tested on systems with Intel Pentium III and Pentium IV Xeon processors. The programs execute reasonably quickly on all of these systems. Even with integer*8 and real*8 variables these programs can run into arithmetic overflow problems, thus limiting the range of sample sizes for which critical values and probabilities can be generated.

References

Bradley, J. V. (1978). Robustness? *British Journal of Mathematical and Statistical Psychology*, *31*, 144-152.

Fix, E. and Hodges, J. L. Jr. (1955). Significance probabilities of the Wilcoxon test. *Annals of Mathematical Statistics*, *26*, 301-312.

Jacobson, J. E. (1963). The Wilcoxon two-sample statistic: Tables and bibliography. *Journal of the American Statistical Association*, *58*, 1086-1103.

Kim, P. J. (1969). On the exact and approximate sampling distribution of the two sample Kolmogorov–Smirnov criterion D_{mn}, $m \leq n$. *Journal of the American Statistical Association, 64*: 1625–1637.

Kim, P. J. & Jennrich, R. I. (1970, 1973). Tables of the exact sampling distribution of the two-sample Kolmogorov–Smirnov criterion, D_{mn}, $m \leq n$. *Selected Tables in Mathematical Statistics, Volume I* (1970, 2nd printing with revisions, 1973), 77-170. Harter, H. L. & Owen, D. B., coeditors, Providence, RI: American Mathematical Society (edited by the Institute of Mathematical Statistics).

Mann, H. B. & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, *18*, 50-60.

Massey, F. J. Jr. (1951). The distribution of the maximum deviation between two sample cumulative step functions. *Annals of Mathematical Statistics*, *22*, 125-128.

Neave, H. R. & Worthington, P. L. B. (1988). *Distribution-free tests*. Unwin Hyman Ltd.

Rosenbaum, S. (1953). Tables for a nonparametric test of dispersion. *Annals of Mathematical Statistics*, *24*, 663-668.

Rosenbaum, S. (1954). Tables for a nonparametric test of location. *Annals of Mathematical Statistics*, *25*, 146-150.

Rosenbaum, S. (1965). On some two-sample non-parametric tests. *Journal of American Statistical Association*, *60*, 1118-1126.

Smirnov, N. V. (1939). Estimating the deviation between the empirical distribution functions of two independent samples. *Bulletin de l'Universite' de Moscou*, *2*(2,3).

Smirnov, N. V. (1948). Table for estimating the goodness of fit of empirical distributions. *Annals of Mathematical Statistics*, *19*, 279–281.

Tukey, J. W. (1959). A quick, compact, two-sample test to Duckworth's specifications. *Technometrics, 1*(1), 31-48.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *I*(6), 80-83.

Wilcoxon, F. (1946). Individual comparisons of grouped data by ranking methods. *Journal of Economic Entomology*, *39*(2), 269.

Wilcoxon, F. (1947). Probability levels for individual comparisons by ranking methods. *Biometrics*, *3*, 119-122.

Wilcoxon, F. & Wilcox, R. A. (1964). Some rapid approximate statistical procedures. Pearl River, NY: Lederle Laboratories Division, American Cyanamid Company. (Originally prepared and distributed in cooperation with the Department of Statistics, The Florida State University, Tallahassee, FL. and revised, 1968).

Wilcoxon, F., Katti, S. K., & Wilcox, R. A. (1970, 1973). Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test. *Selected Tables in Mathematical Statistics*, *Volume I* (1970, 2nd printing with revisions, 1973), 171-259. Harter, H. L. & Owen, D. B., coeditors, Providence, RI: American Mathematical Society (edited by the Institute of Mathematical Statistics).

Wilks, S. S. (1942). Statistical prediction with special reference to the problem of tolerance limits. *Annals of Mathematical Statistics*, *13*, 400-409.

Programs

Tukey's (1959) Two-sample Test to Duckworth's Specifications (Tukey's Quick Test)

Main program for printing tables

```
! ***************************************************************************
! program:  CVTQTJ.exe
! source:   CVTQTJ.f90
! author:   Bruce R. Fay
! date:     17 Oct 2002 17:32 EDT
! purpose:  Test harness for critical value modules for Tukey's Quick
!           test of location to Duckworth's specifications
! desc:     Prints tables of critical values with associated probabilities.
! ***************************************************************************
program CVTQTJ
use CVTQTJmod
implicit none
! DECLARE LOCAL VARIABLES
integer :: i, j, LU1, LU2, ios, testnum
integer, dimension(:) :: CVi(4)
real*8, dimension(:) :: PVr(4)
! GET USER INPUTS
write(*,*) "Program CVTQTJ.exe by Bruce R. Fay"
write(*,*) "Critical values for Tukey's Quick Test"
write(*,*)
write(*,*) "Creates output files CVTQTJbc_.txt and CVTQTJbf_.txt"
write(*,*) "in current directory."
write(*,*)
write(*,*) "Select one of the following:"
write(*,*)
write(*,*) " 0 - to exit program"
write(*,*) " 1 - to generate CV/PV tables"
write(*,*)
Do
  read(*,*) testnum
  If ( (testnum >= 0).and.(testnum <= 1) ) EXIT
  write(*,*) "enter 0 to exit, 1 to run"
End Do
If (testnum == 0) GOTO 9999  ! check for user termination
! OPEN FILES FOR OUTPUT
LU1 = 8
open(unit=LU1, file='CVTQTJbc_.txt', iostat=ios)
IF (ios > 0 ) then
  write(*,*) "Error opening file 'CVTQTJbc_.txt' "
  GOTO 9999
End if
LU2 = 9
open(unit=LU2, file='CVTQTJbf_.txt', iostat=ios)
IF (ios > 0 ) then
  write(*,*) "Error opening file 'CVTQTJbf_.txt' "
  GOTO 9999
End if
! DEFINE OUTPUT FORMATS
100 format(" 1-tailed CVs at stated alpha levels")
200 format(" n1 n2 - .05 - -.025 - - .01 - -.005 - | &
            & - .05 - -.025 - - .01 - -.005 -")
```

```
300 format(2I3,4I8,3x,4F8.4)
! CREATE BEST-CONSERVATIVE TABLES
write(LU1,*) "Program CVTQTJ by Bruce R. Fay"
write(LU1,*)
write(LU1,*) "Tukey's quick test of location for two independent samples,"
write(LU1,*) "best-conservative critical values generated based on"
write(LU1,*) "Tukey (1959) using CVTQTJbc() in CVTQTJmod."
write(LU1,*)
call CV_TQTJbc_init     ! generate the BC CV/PV tables
write(LU1,100)          ! print header information
write(LU1,*)
write(LU1,200)          ! print column headers for this format
write(LU1,*)
Do i = 1,30             ! output the tables to file
  Do j = i,30
    call CV_TQTJbc(i,j,CVi,PVr)
    write(LU1,300) i,j,CVi(1:4),PVr(1:4)
  End Do
  write(LU1,*)
End Do
! CREATE BEST-FIT TABLES
write(LU2,*) "Program CVTQTJ by Bruce R. Fay"
write(LU2,*)
write(LU2,*) "Tukey's quick test of location for two independent samples."
write(LU2,*) "Best-fitting critical values generated based on Tukey (1959)"
write(LU2,*) "using CVTQTJbf() in CVTQTJmod, where best-fit is defined as"
write(LU2,*) "pr <= alpha + 10% when this probability is closer to alpha"
write(LU2,*) "than the first available CV with pr < alpha."
write(LU2,*)
call CV_TQTJbf_init     ! generate the BF CV/PV tables
write(LU2,100)          ! print header information
write(LU2,*)
write(LU2,200)          ! print column headers for this format
write(LU2,*)
Do i = 1,30             ! output the tables to file
  Do j = i,30
    call CV_TQTJbf(i,j,CVi,PVr)
    write(LU2,300) i,j,CVi(1:4),PVr(1:4)
  End Do
  write(LU2,*)
End Do
! CLOSE FILES
close(unit=LU1, status='keep', iostat=ios)
If (ios > 0) then
  write(*,*) "Error closing file 'CVTQTJbc_.txt' "
End If
close(unit=LU2, status='keep', iostat=ios)
If (ios > 0) then
  write(*,*) "Error closing file 'CVTQTJbf_.txt' "
End If
9999 stop
end program CVTQTJ
```

Module for generating critical values and probabilities

```
! ***********************************************************************
! module:   CVTQTJmod
! source:   CVTQTJmod.f90
! based on: Tukey (1959) A quick, compact, two-sample test to Duckworth's
!           specifications, Technometrics Vol. 1 No. 1 (Feb) pgs.31-48,
!           method for generating exact critical values.
! author:   Bruce R. Fay
! date:     17 Oct 2002 19:03 EDT
! purpose:  Provide the exact critical values for Tukey's Quick Test for
!           2-independent-samples, both best-conservative and best-fit.
! desc:     Generates the CVTs and PVTs on initialization and provides
!           an entry point that returns up to four critical values based
!           on the incoming values of n1 and n2.  Checks are made that
!           n1, n2 are in the appropriate range and relationship for the
!           tables with 1 <= n1 <= n2 <= 30.
! Notes:    Best-conservative values are those for which pr(h) <= nominal
!           alpha.  Best-fit CVs are generated by the same method but with
!           pr(h) <= alpha+10% if pr(h+1) < alpha and is further from alpha
!           than pr(h).
! ***********************************************************************
module CVTQTJmod
implicit none
private
public :: CV_TQTJbc_init, CV_TQTJbc, CV_TQTJbf_init, CV_TQTJbf, N_c_m
contains
! ***********************************************************************
subroutine CV_TQTJbc_init
! INTERFACE
! There are no arguments for CV_TQTJbc_init.  The calling routine must call
! this subroutine once to build the CV and PV tables prior to calling
! CV_TQTJbc() to obtain critical values for specific n1, n2.  Calling routine
! must also declare an integer vector of length 4 and a real*8 vector of
! length 4 and pass them into receive the critical values and their
! associated probability values.  For entry CV_TQTJbc(s1,s2,CV,PV):
!     s1    ::    sample size for 1st group ( <= s2 )
!     s2    ::    sample size for 2nd group
!     CV    ::    critical values vector (length 4)
!     PV    ::    probability values vector (length 4)
! DECLARE DUMMY VARIABLES
integer, intent(in) :: s1, s2
integer, intent(out), dimension(:) :: CV
real*8, intent(out), dimension(:) :: PV
! DESCRIPTION
! At entry CV_TQTJbc(), for s1 <= s2, returns up to four critical values,
! if available, in vector CV(:), as follows:
!     CV(1) = 1-tailed alpha .05  (2-tailed alpha .10)
!     CV(2) = 1-tailed alpha .025 (2-tailed alpha .05)
!     CV(3) = 1-tailed alpha .01  (2-tailed alpha .02)
!     CV(4) = 1-tailed alpha .005 (2-tailed alpha .01)
! The actual 1-tailed probabilities corresponding to the above CVs are
! returned in PV(1:4).  If a critical value is not available, a -1 is
! returned instead, with associated probability zero.  Critical values may
! not be available because s1 and s2 are a) too small, b) too large, or
```

```
! c) too different.  Unequal s1, s2 are supported for 1 <= s1 <= s2 <= 30.
! DECLARE LOCAL VARIABLES
integer :: h, n1, n2, v1, v2, w1, w2
integer (kind=8) :: wv1, wv2
integer (kind=8), dimension(30,30), save :: CVTbc05, CVTbc025
integer (kind=8), dimension(30,30), save :: CVTbc01, CVTbc005
integer (kind=8), dimension(0:30,1:30), save :: Atbl
integer (kind=8) :: comb, A1, A2, Adiff
integer (kind=8), parameter :: zero=0, one=1, two=2
real (kind=8), dimension(30,30), save :: PVTbc05, PVTbc025, PVTbc01, PVTbc005
real (kind=8), parameter :: m05=0.050, m025=0.025, m01=0.01, m005=0.005
real (kind=8) :: c05, c025, c01, c005, rcomb, rdiff
logical :: fnd05, fnd025, fnd01, fnd005
! Build the A table
!
!        Column(w)
!           -2    -1     0     1     2     3     4    ...   30
!        ---------------------------------------------------------
!     -1|    0     0 |   0 |   0     0     0      0   ...    0
!      0|    0     0 |   0 |   0     0     0      0   ...    0
!       |------------+-----+----------------------------------
!      1|    0     1 |   1 |   1     1     1      1   ...    1
!       |------------+-----+----------------------------------
! Row  2|    1     1 |   2 |   3     4     5      6   ...   32
! (v)  3|    1     2 |   4 |   7    11    16     22   ...    .
!      4|    2     4 |   8 |  15    26    42     64   ...    .
!      5|    4     8 |  16 |  31    57    99    163   ...    .
!      6|    8    16 |  32 |   .     .     .      .   ...    .
!      .|   16    32 |   . |   .
!      .|   32     . |   . |
!      .|    .     . |   . |
!     30|    .      536870912
!
! Note: The A table is only built for columns 0 to 30 and rows 1 to 30.  All
! entries for rows less than one are zero and all entries for columns
! less than zero (with rows of 1 or more) can be determined by direct
! formula (see code).
Atbl(0:30,1) = one      ! first row, all columns, entries = 1
Do v1 = 2,30            ! first (zero) column, row entries are 2^(row-1)
  Atbl(0,v1) = two**(v1-1)
End Do
Do v1 = 2,30            ! previous column same row + same column previous row
  Do w1 = 1,30
      Atbl(w1,v1) = Atbl(w1-1,v1) + Atbl(w1,v1-1)
  End Do
End Do
CVTbc05 = -1   ! initialize the CV tables to -1 (indicates no valid entry)
CVTbc025 = -1
CVTbc01 = -1
CVTbc005 = -1
PVTbc05 = 0.0  ! initialize the PV tables to 0.0 (indicates no valid entry)
PVTbc025 = 0.0
PVTbc01 = 0.0
PVTbc005 = 0.0
! Determine the critical values and associated actual probabilities
Do n1 = 1,30            ! n1 for CV/PV tables
  Do n2 = n1,30         ! n2 for CV/PV tables
```

```
    fnd05 = .false.      ! reset found flags for each alpha level
    fnd025 = .false.
    fnd01 = .false.
    fnd005 = .false.
    comb = N_c_m(n1,n2) ! get the number of combinations for n1 and n2
    rcomb = real(comb)
    c05 = rcomb * m05   ! calculate the comparison values for each alpha
    c025 = rcomb * m025
    c01 = rcomb * m01
    c005 = rcomb * m005
    Do h = 1,(n1+n2)    ! h will be the CV if/when we find the right one
      w1 = n2-h               ! Find A1 as Atbl(n2-h,n1)
      v1 = n1                 ! since n1 >= 1, v is a valid row for Atbl
      wv1 = w1 + v1           ! = n1 + n2 - h
      If (w1 >= 0) then       ! it's OK to use the Atbl to get A1
        A1 = Atbl(w1,v1)
      Else                    ! calculate A1 by formula
        If (wv1 > 0) then           ! w < 0,  v > 0,  |v| > |w|
          A1 = two**(wv1-1)
        Else If (wv1 == 0) then     ! w = -v
          A1 = one
        Else If (wv1 < 0 ) then     ! w < 0,  v > 0,  |v| < |w|
          A1 = zero
        End If
      End If
      v2 = n1-h               ! Find A2 as Atbl(n2,n1-h)
      w2 = n2                 ! since n2 >= 1, w is a valid column for Atbl
      If(v2 >= 1) then        ! valid row for Atbl
        A2 = Atbl(w2,v2)
      Else
        A2 = zero
      End If
      Adiff = A1 - A2
      rdiff = real(Adiff)
      If ( (rdiff <= c05).and.(.not.fnd05) ) then
        CVTbc05(n1,n2) = h
        PVTbc05(n1,n2) = rdiff/rcomb
        fnd05 = .true.
      End If
      If ( (rdiff <= c025).and.(.not.fnd025) ) then
        CVTbc025(n1,n2) = h
        PVTbc025(n1,n2) = rdiff/rcomb
        fnd025 = .true.
      End If
      If ( (rdiff <= c01).and.(.not.fnd01) ) then
        CVTbc01(n1,n2) = h
        PVTbc01(n1,n2) = rdiff/rcomb
        fnd01 = .true.
      End If
      If ( (rdiff <= c005).and.(.not.fnd005) ) then
        CVTbc005(n1,n2) = h
        PVTbc005(n1,n2) = rdiff/rcomb
        fnd005 = .true.
      End If
      If (fnd05.and.fnd025.and.fnd01.and.fnd005) exit
    End Do
  End Do
```

```
End Do
Return
! ---------------------------------------------------------------------------
entry CV_TQTJbc(s1,s2,CV,PV)
CV(:) = -1        ! initialize all return CVs to 'not available'
PV(:) = 0.0       ! initialize all return PVs to 'not available'
If ((1<=s1).and.(s1<=30).and.(1<=s2).and.(s2<=30).and.(s1<=s2)) then
  CV(1) = CVTbc05(s1,s2)
  CV(2) = CVTbc025(s1,s2)
  CV(3) = CVTbc01(s1,s2)
  CV(4) = CVTbc005(s1,s2)
  PV(1) = PVTbc05(s1,s2)
  PV(2) = PVTbc025(s1,s2)
  PV(3) = PVTbc01(s1,s2)
  PV(4) = PVTbc005(s1,s2)
End If
Return
! --------------------------------------------------------------------------
end subroutine CV_TQTJbc_init
! ***************************************************************************
subroutine CV_TQTJbf_init
! see subroutine CV_TQTJbc_init above for documentation and comments
! DECLARE DUMMY VARIABLES
integer, intent(in) :: s1, s2
integer, intent(out), dimension(:) :: CV
real*8, intent(out), dimension(:) :: PV
! DECLARE LOCAL VARIABLES
integer :: h, n1, n2, v1, v2, w1, w2
integer (kind=8) :: CV1tmp, CV2tmp, CV3tmp, CV4tmp, wv1, wv2
integer (kind=8), dimension(30,30), save :: CVTbf05, CVTbf025
integer (kind=8), dimension(30,30), save :: CVTbf01, CVTbf005
integer (kind=8), dimension(0:30,1:30), save :: Atbl
integer (kind=8) :: comb, A1, A2, Adiff
integer (kind=8), parameter :: two=2
real (kind=8), dimension(30,30), save :: PVTbf05, PVTbf025, PVTbf01, PVTbf005
real (kind=8), parameter :: m05=0.05, m025=0.025, m01=0.01, m005=0.005
real (kind=8), parameter :: m055=0.055, m0275=0.0275
real (kind=8), parameter :: m011=0.011, m0055=0.0055
real (kind=8) :: c05, c025, c01, c005, c055, c0275, c011, c0055, rcomb, rdiff
real (kind=8) :: ptmp, PV1tmp, PV2tmp, PV3tmp, PV4tmp
logical :: fnd05, fnd025, fnd01, fnd005
! BUILD THE A TABLE
Atbl(0:30,1) = 1  ! first row
Do v1 = 1,30      ! first column
  Atbl(0,v1) = two**(v1-1)
End Do
Do v1 = 2,30      ! previous column same row + same column previous row
  Do w1 = 1,30
    Atbl(w1,v1) = Atbl(w1-1,v1) + Atbl(w1,v1-1)
  End Do
End Do
CVTbf05 = -1   ! initialize the CV tables to -1 (indicates no valid entry)
CVTbf025 = -1
CVTbf01 = -1
CVTbf005 = -1
PVTbf05 = 0.0  ! initialize the PV tables to 0.0 (indicates no valid entry)
PVTbf025 = 0.0
```

```
PVTbf01 = 0.0
PVTbf005 = 0.0
! Determine the critical values and associated actual probabilities
Do n1 = 1,30
  Do n2 = n1,30
    fnd05 = .false.  ! reset found flags for each alpha level
    fnd025 = .false.
    fnd01 = .false.
    fnd005 = .false.
    comb = N_c_m(n1,n2)  ! get the number of combinations for n1 and n2
    rcomb = real(comb)
    c05 = rcomb * m05    ! calculate the comparison values for each alpha
    c025 = rcomb * m025
    c01 = rcomb * m01
    c005 = rcomb * m005
    c055 = rcomb * m055  ! comparison values for alpha + 10%
    c0275 = rcomb * m0275
    c011 = rcomb * m011
    c0055 = rcomb * m0055
    PV1tmp = 1.0  ! initialize temporary probability values
    PV2tmp = 1.0
    PV3tmp = 1.0
    PV4tmp = 1.0
    Do h = 1,(n1+n2)
      w1 = n2-h
      v1 = n1
      wv1 = w1 + v1
      If (w1 >= 0) then
        A1 = Atbl(w1,v1)
      Else
        If (wv1 > 0) then
          A1 = 2**(wv1-1)
        Else If (wv1 == 0) then
          A1 = 1
        Else If (wv1 < 0) then
          A1 = 0
        End If
      End If
      w2 = n2
      v2 = n1-h
      If (v2 >= 1) then
        A2 = Atbl(w2,v2)
      Else
        A2 = 0
      End If
      Adiff = A1 - A2
      rdiff = real(Adiff)
      If((c05 < rdiff).and.(rdiff <= c055).and.(.not.fnd05)) then
        CV1tmp = h
        PV1tmp = rdiff/rcomb
      Else If((rdiff <= c05).and.(.not.fnd05)) then
        ptmp = rdiff/rcomb
        If((.05 - ptmp) <= (PV1tmp - .05)) then
          CVTbf05(n1,n2) = h
          PVTbf05(n1,n2) = ptmp
        Else
          CVTbf05(n1,n2) = CV1tmp
```

```
          PVTbf05(n1,n2) = PV1tmp
        End If
        fnd05 = .true.
      End If
      If((c025 < rdiff).and.(rdiff <= c0275).and.(.not.fnd025)) then
        CV2tmp = h
        PV2tmp = rdiff/rcomb
      Else If((rdiff <= c025).and.(.not.fnd025)) then
        ptmp = rdiff/rcomb
        If((.025 - ptmp) <= (PV2tmp - .025)) then
          CVTbf025(n1,n2) = h
          PVTbf025(n1,n2) = ptmp
        Else
          CVTbf025(n1,n2) = CV2tmp
          PVTbf025(n1,n2) = PV2tmp
        End If
        fnd025 = .true.
      End If
      If((c01 < rdiff).and.(rdiff <= c011).and.(.not.fnd01)) then
        CV3tmp = h
        PV3tmp = rdiff/rcomb
      Else If((rdiff <= c01).and.(.not.fnd01)) then
        ptmp = rdiff/rcomb
        If((.01 - ptmp) <= (PV3tmp - .01)) then
          CVTbf01(n1,n2) = h
          PVTbf01(n1,n2) = ptmp
        Else
          CVTbf01(n1,n2) = CV3tmp
          PVTbf01(n1,n2) = PV3tmp
        End If
        fnd01 = .true.
      End If
      If((c005 < rdiff).and.(rdiff <= c0055).and.(.not.fnd005)) then
        CV4tmp = h
        PV4tmp = rdiff/rcomb
      Else If((rdiff <= c005).and.(.not.fnd005)) then
        ptmp = rdiff/rcomb
        If((.005 - ptmp) <= (PV4tmp - .005)) then
          CVTbf005(n1,n2) = h
          PVTbf005(n1,n2) = ptmp
        Else
          CVTbf005(n1,n2) = CV4tmp
          PVTbf005(n1,n2) = PV4tmp
        End If
        fnd005 = .true.
      End If
      If (fnd05.and.fnd025.and.fnd01.and.fnd005) exit
    End Do
  End Do
End Do
Return
! ----------------------------------------------------------------------------
entry CV_TQTJbf(s1,s2,CV,PV)
CV(:) = -1        ! initialize all return CVs to 'not available'
PV(:) = 0.0       ! initialize all return PVs to 'not available'
If ((1<=s1).and.(s1<=30).and.(1<=s2).and.(s2<=30).and.(s1<=s2)) then
  CV(1) = CVTbf05(s1,s2)
```

```
  CV(2) = CVTbf025(s1,s2)
  CV(3) = CVTbf01(s1,s2)
  CV(4) = CVTbf005(s1,s2)
  PV(1) = PVTbf05(s1,s2)
  PV(2) = PVTbf025(s1,s2)
  PV(3) = PVTbf01(s1,s2)
  PV(4) = PVTbf005(s1,s2)
End If
Return
! ----------------------------------------------------------------------------
end subroutine CV_TQTJbf_init
!*****************************************************************************
function N_c_m(a,b) result(F)
! Calculates number of combinations, 'N chose m' or nCm where
! N = a+b and m = a (equivalent to m = b).  The formula is
! N!/(m!(N-m)!) = (a+b)!/(a!b!) =
! [1*2*...*b*(b+1)*...*(a+b)]/[(1*2*...*a)*(1*2*...*b))]
! This is equivalent to [(b+1)(b+2)...(b+a)]/[a!] or
! [(b+1)(b+2)...(b+a)]/[1*2*...*a], which is implemented here.
! This computation is particularly efficient if a <= b, as it is in
! subroutines CV_TQTJbc_init and CV_TQTJbf_init above.  Both a and b must
! be >= zero, otherwise the function returns with value -1 to indicate an
! error.
! DECLARE DUMMY VARIABLES
integer, intent(in) :: a, b
! DECLARE LOCAL VARIABLES
integer :: i
integer (kind=8) :: C, F, num
! VARIABLE DEFINTIONS
!     a     ::    number of items in first group
!     b     ::    number of items in second group
!     C     ::    accumulator for number of combinations
!     F     ::    function result
!     i     ::    loop variable
!     num   ::    numerator factor for combinations computation
If((a>=0).and.(b>=0)) then    ! both inputs non-negative
  If((a>=1).and.(b>=1)) then  ! both inputs > 0, proceed
    C = 1
    Do i = 1,a
      num = i + b
      C = (C * num) / i
    End Do
  Else      ! both inputs zero or one positive and one zero
    C = 1
  End If
Else  ! at least one negative input
  C = -1  ! error
End If
F = C
return
end function N_c_m
! ***************************************************************************
end module CVTQTJmod
```

Rosenbaum's Test of Location

Main program for printing tables

```
! ***************************************************************************
! program:  CVRBTJ
! source:   CVRBTJ.f90
! based on: CVRBT.f90 as of 29 Apr 2002 15:22 EDT
! author:   Bruce R. Fay
! date:     18 Oct 2002 18:13 EDT
! purpose:  Generate and print critical value table for Rosenbaum's test
!           of location for 2 independent samples.
! ***************************************************************************
program CVRBT
use CVRBjmod
implicit none
! DECLARE VARIABLES
integer :: i, j, LU, ios, testnum
integer, dimension(:) :: CVi(4
real*8, dimension(:) :: PVr(4)
! DEFINE FORMATS FOR OUTPUT FILE
100 format(" 1-tailed CVs at stated alpha levels")
200 format("       | - - - - - - -   CV  - - - - - - -  |  &
    &- - - - - - -  PV  - - - - - - -  |")
300 format(" n1 n2 - .05 - - .025- - .01 - - .005-     &
    &- .05 - - .025- - .01 - - .005-")
400 format(2I3,4I8,4x,4F8.4)
! GET USER INPUTS
write(*,*) "Program CVRBTJ.exe by Bruce R. Fay"
write(*,*) "Generate best conservative critical values and associated"
write(*,*) "probabilities for Rosenbaum's Test for two-independent-samples"
write(*,*) "and output results to file"
write(*,*)
write(*,*) "Select one of the following:"
write(*,*)
write(*,*) " 0 - to exit program"
write(*,*) " 1 - to generate values"
write(*,*)
Do
  read(*,*) testnum
  If ( (testnum >= 0).and.(testnum <= 1) ) then
    EXIT
  Else
    write(*,*) "enter 0 - 4 please"
  End if
End Do
If (testnum == 0) GOTO 9999  ! check for user termination
! OPEN OUTPUT FILE AND WRITE FILE HEADER
LU = 8
open(unit=LU, file='CVRBTJ_.txt', iostat=ios)
IF (ios > 0 ) then
  write(*,*) "Error opening file 'CVRBTJ_.txt' "
  GOTO 9999
End if
write(LU,*) "Program CVRBTJ.exe by (Author's name here)"
write(LU,*) "Output file CVRBTJ_.txt"
write(LU,*)
```

```
write(LU,*) "Generate best conservative critical values and associated"
write(LU,*) "probabilities for Rosenbaum's Test for two-independent-samples"
write(LU,*) "based on formula in Neave & Worthington (1988)"
write(LU,*) "Distribution-free Tests, p. 148"
write(LU,*)
write(LU,*) "n1 = m, n2 = n, n1 is the size of the sample from which"
write(LU,*) "the test statistic is calculated (length of extreme run)"
write(LU,*)
! GENERATE VALUES AND OUTPUT TO FILE
call CV_RBJ_init
write(LU,100)  ! print header information
write(LU,*)
write(LU,200)  ! print column headers for this format
write(LU,300)
write(LU,*)
Do i = 1,50
  Do j = 1,50
    call CV_RBJbc(i,j,CVi,PVr)
    write(LU,400) i,j,CVi(1:4),PVr(1:4)
  End Do
  write(LU,*)
End Do
! CLOSE FILE
close(unit=LU, status='keep', iostat=ios)
If (ios > 0) then
  write(*,*) "Error closing file 'CVRBTJ_.txt' "
End If
9999 stop
end program CVRBT
```

Module for generating critical values and probabilities

```
! ***************************************************************************
! module:   CVRBJmod
! source:   CVRBJmod.f90
! based on: CVRB4mod.f90 as of 20 Apr 2002 23:01 EDT and
!           Neave & Worthington (1988) Distribution-free Tests, Table J,
!           383-386 and Rosenbaum (1954) Tables for a nonparametric
!           test of location, Annals of Mathematical Statistics, Vol. 25,
!           146-150.  The later tables also appear in Owen (1962)
!           Handbook of Statistical Tables, 499-503.
! author:   Bruce R. Fay
! date:     18 Oct 2002 18:12 EDT
! purpose:  Provide the critical values for Rosenbaum's Test of Location
!           for 2-independent-samples based on the method of Neave &
! desc:     Worthington (1988) p. 148 to calculate probability of a run of h
!           values from sample m out of a combined sample of N = m + n.  The
!           formula is
!           m!(N-h)!/[N!(m-h)!] = m/N x (m-1)/(N-1) x ... x (m-h+1)/(N-m+1)
!           The value of h associated with the largest such probability that
!           is <= nominal alpha is the critical value for that situation.
!           Thus all CVs are BEST CONSERVATIVE with pr(CV) <= nominal alpha.
!           Creates the CVTs and PVTs on initialization and provides an
!           entry point that returns up to 4 critical values, and their
!           associated probabilities, based on the incoming values of m
!           and n.  Checks are made that m and n are in the appropriate
```

```
!           ranges, 1 <= m <= n and 1 <= n <= 50.  The sample from which
!           the statistic is calculated must have sample size m.
!***************************************************************************
module CVRBJmod
implicit none
private
public :: CV_RBJ_init, CV_RBJbc
contains
! **************************************************************************
subroutine CV_RBJ_init
! INTERFACE
! There are no arguments for CV_RBJ_init.  The calling routine must call this
! subroutine once to build the CV and PV tables prior to calling CV_RBJbc()
! to obtain critical values and associated probabilities for specific n1, n2.
! The calling routine must declare an integer vector of length 4 and a real*8
! vector of length 4 and pass them in as arguments to receive the critical
! values and their associated probabilities.  For entry CV_RBJbc(m,n,CV,PV):
!    m   ::  sample size for group from which the statistic is calculated
!    n   ::  sample size for the other group
!    CV  ::  critical values vector (integer, length 4)
!    PV  ::  probability values vector (real, length 4)
! Unequal n1, n2 are supported for all n1, n2, both <= 50, where m is the
! sample size of the sample from which the statistic is calculated, i.e.,
! the sample with the global maximum.
! DESCRIPTION
! At entry CV_RBJ(), returns up to four critical values, if available, in
! vector CV(:), as follows:
!    CV(1) = 1-tailed alpha .05  (2-tailed alpha .10)
!    CV(2) = 1-tailed alpha .025 (2-tailed alpha .05)
!    CV(3) = 1-tailed alpha .01  (2-tailed alpha .02)
!    CV(4) = 1-tailed alpha .005 (2-tailed alpha .01)
! If a critical value is not available, a -1 is returned instead with
! associated probability 0.  Critical values may not be available because
! n1 and n2 are a) too small, b) too large, or c) too different.
! DECLARE DUMMY VARIABLES
integer, intent(in) :: m, n
integer, intent(in out), dimension(:) :: CV
real*8, intent(in out), dimension(:) :: PV
! DECLARE LOCAL VARIABLES
integer, dimension(50,50), save :: CVTbc1, CVTbc2, CVTbc3, CVTbc4
integer :: h, mm, nn, mn
real*8, dimension(50,50), save :: PVTbc1, PVTbc2, PVTbc3, PVTbc4
real*8 :: R, rm, T
logical :: p05, p025, p01, p005
CVTbc1 = -1  ! initialize the CV tables to -1 (indicates no valid entry)
CVTbc2 = -1
CVTbc3 = -1
CVTbc4 = -1
PVTbc1 = 0.0  ! initialize the PV tables to 0 (indicates no valid entry)
PVTbc2 = 0.0
PVTbc3 = 0.0
PVTbc4 = 0.0
Do nn = 1,50  ! generate the CV and PV tables
  Do mm = 1,50
    p05 = .false.
    p025 = .false.
    p01 = .false.
```

```
    p005 = .false.
    mn = mm + nn
    T = real(mn)
    rm = real(mm)
    R = 1.0
    Do h = 1,mm
      R = R * rm / T
      rm = rm - 1.0
      T = T - 1.0
      If( (R <= .05).and.(.not.p05) ) then
        CVTbc1(mm,nn) = h
        PVTbc1(mm,nn) = R
        p05 = .true.
      End If
      If( (R <= .025).and.(.not.p025) ) then
        CVTbc2(mm,nn) = h
        PVTbc2(mm,nn) = R
        p025 = .true.
      End If
      If( (R <= .01).and.(.not.p01) ) then
        CVTbc3(mm,nn) = h
        PVTbc3(mm,nn) = R
        p01 = .true.
      End If
      If( (R <= .005).and.(.not.p005) ) then
        CVTbc4(mm,nn) = h
        PVTbc4(mm,nn) = R
        p005 = .true.
      End If
      If (p05.and.p025.and.p01.and.p005) exit
    End Do
  End Do
End Do
return
! ----------------------------------------------------------------------------
entry CV_RBJbc(m,n,CV,PV)
! CV_RBJbc() must be called with m = sample size of group from which the
!  statistic is calculated (group with global maximum value).
CV(:) = -1        ! initialize all return CVs to 'not available'
PV(:) = 0.0       ! initialize all return PVs to 'not available'
If ((m >= 1).and.(m <= 50).and.(n >= 1).and.(n <= 50)) then
  CV(1) = CVTbc1(m,n)
  CV(2) = CVTbc2(m,n)
  CV(3) = CVTbc3(m,n)
  CV(4) = CVTbc4(m,n)
  PV(1) = PVTbc1(m,n)
  PV(2) = PVTbc2(m,n)
  PV(3) = PVTbc3(m,n)
  PV(4) = PVTbc4(m,n)
End If
return
! ----------------------------------------------------------------------------
end subroutine CV_RBJ_init
! ****************************************************************************
end module CVRBJmod
```

Kolmogorov-Smirnov Test of General Differences

Main program for printing tables

```
! ***************************************************************************
! program:  CVKSTJ
! source:   CVKSTJ.f90
! based on: CVKST.f90 as of 29 Apr 2002 15:10 EDT
! author:   Bruce R. Fay
! date:     19 Oct 2002 10:59 EDT
! purpose:  Test harness for critical value modules for Kolmogorov-
!           Smirnov 2-independent-samples test for general differences.
! desc:     Provides user choice of printing critical values and
!           associated probability values for 2-sided tests based on ABS(Dmn)
!           or for 1-sided tests based on either on Dneg or Dpos.  Module
!           CVKSJmod generates the 2-sided values.
! ***************************************************************************
program CVKSTJ
use CVKSJmod
implicit none
! DECLARE VARIABLES
integer :: i, j, k, LU, ios, testnum
integer, dimension(:) :: CVi(4)
real, dimension(:) :: PVr(4)
! GET USER INPUTS
write(*,*) "Program CVKSTJ.exe by Bruce R. Fay"
write(*,*) "Kolmogorov-Smirnov test of general differences for"
write(*,*) "two independent samples - critical value tables with"
write(*,*) "probabilities"
write(*,*)
write(*,*) "Select one of the following:"
write(*,*)
write(*,*) " 0 - exit"
write(*,*) " 1 - generate 1-tailed CVs and actual p values using CVKSJmod"
write(*,*) " 2 - generate 2-tailed CVs and actual p values using CVKSJmod"
write(*,*)
Do
  read(*,*) testnum
  If ( (0 <= testnum).and.(testnum <= 2) ) EXIT
  write(*,*) "enter 0 - 2 please"
End Do
If (testnum == 0) GOTO 9999  ! check for user termination
! OPEN OUTPUT FILE AND WRITE FILE HEADER
LU = 8
open(unit=LU, file='CVKSTJ_.txt', iostat=ios)
IF (ios > 0 ) then
  write(*,*) "Error opening file 'CVKSTJ_.txt' "
  GOTO 9999
End if
write(LU,*) "Program CVKSTJ by (Author's name goes here)"
write(LU,*) "File CVKSTJ_.txt"
write(LU,*)
! DEFINE FORMATS FOR OUTPUT FILE
100 format(" 2-tailed CVs and PVs at stated alpha levels")
110 format(" 1-tailed CVs and PVs at stated alpha levels")
120 format("        ---- nominal alpha 2-tailed ---   &
```

```
           &-------- actual 2-tailed prob ---------")
130 format("  n1 n2 - .10 - - .05 - - .02 - - .01 -  &
              & -- .10 -- -- .05 -- -- .02 -- -- .01 --")
140 format("        ---- nominal alpha 1-tailed ---   &
           &-------- actual 1-tailed prob ---------")
150 format("  n1 n2 - .05 - - .025  - .01 - - .005   &
              & -- .05 -- -- .025 - -- .01 -- -- .005 -")
160 format(1x,2I3,4I8,2x,4F10.6)
Select Case(testnum)
Case(1)
  write(*,*) "Outputing CVT to file for K-S 2-i-s t-g-d"
  write(*,*) "generated CVs based on Kim & Jennrich, with"
  write(*,*) "actual 1-tailed probabilities"
  write(*,*)
  write(LU,*) "Kolmogorov-Smirnov test of general differences for"
  write(LU,*) "two independent samples, critical values based on"
  write(LU,*) "Kim & Jennrich (1970,1973), with actual 1-tailed"
  write(LU,*) "probabilities generated by CVKSJmod"
  write(LU,*)
  write(*,*) "Generating CV tables"
  call CV_KSJ_init
  write(*,*) "CV_KSJ_init completed - CV tables built"
  write(LU,110)  ! print header information
  write(LU,*)
  write(LU,140)  ! print column headers for this format
  write(LU,150)
  write(LU,*)
  Do j = 1,50
    Do i = 1,j
      call CV_KSJbc(i,j,CVi,PVr)
      PVr = PVr/2.0
      write(LU,160) i,j,CVi(1:4),PVr(1:4)
    End Do
    write(LU,*)
  End Do
Case(2)  ! 2-sided values w/ actual probabilities
  write(*,*) "Outputing CVT to file for K-S 2-i-s t-g-d"
  write(*,*) "generated CVs based on Kim & Jennrich, with"
  write(*,*) "actual 2-tailed probabilities"
  write(*,*)
  write(LU,*) "Kolmogorov-Smirnov test of general differences for"
  write(LU,*) "two independent samples, critical values based on"
  write(LU,*) "Kim & Jennrich (1970,1973), with actual 2-tailed"
  write(LU,*) "probabilities generated by CVKSJmod"
  write(LU,*)
  write(*,*) "Generating CV tables"
  call CV_KSJ_init
  write(*,*) "CV_KSJ_init completed - CV tables built"
  write(LU,100)  ! print header information
  write(LU,*)
  write(LU,120)  ! print column headers for this format
  write(LU,130)
  write(LU,*)
  Do j = 1,50
    Do i = 1,j
      call CV_KSJbc(i,j,CVi,PVr)
      write(LU,160) i,j,CVi(1:4),PVr(1:4)
```

```
    End Do
    write(LU,*)
  End Do
End Select
! CLOSE FILE
close(unit=LU, status='keep', iostat=ios)
If (ios > 0) then
  write(*,*) "Error closing file 'CVKSTJ_.txt' "
End If
9999 stop
end program CVKSTJ
```

Module for generating critical values and probabilities

```
! ***************************************************************************
! module:   CVKSJmod
! source:   CVKSJmod.f90
! based on: CVKS3mod as of 05 Jun 2002 19:00, which is based on the
!           Kim & Jennrich Tables of the exact sampling distribution of
!           the two-sample Kolmogorov=Smirnov criterion, Dmn, m<=n in
!           Selected Tables in Mathematical Statistics, Vol. 1, 77-170
!           (1970) Harter & Owens (eds) 2nd printing (1973) with revisions,
!           published by American Mathematical Society for the
!           Institute of Mathematical Statistics
! author:   Bruce R. Fay
! date:     19 Oct 2002 10:48 EDT
! purpose:  Provide the best conservative critical values for the
!           Kolmogorov-Smirnov 2-independent-samples test for general
!           differences.
! desc:     Generates the CVTs on initialization and provides an entry
!           point that returns up to 4 critical values based on the
!           incoming values of m and n.  Checks are made that
!           1 <= m <= n <= 50.  If n1, n2 are not in this range and
!           relationship, the lookup is not performed.  When CVs are
!           not available, a value of -1 is returned.
! ***************************************************************************
module CVKSJmod
implicit none
private
public :: CV_KSJ_init, CV_KSJbc
contains
! ***************************************************************************
subroutine CV_KSJ_init
! INTERFACE
! There are no arguments for CV_KSJ_init.  The calling routine must call this
! subroutine once to build the CV table prior to calling CV_KSJbc() to obtain
! critical values and probabilities for specific m and n.  The calling
! routine must also declare an integer vector of length 4 and pass it in to
! receive the critical values as well as a real vector of length 4 and pass
! it in to receive the probabilities.  For entry CV_KSJbc(m,n,CV,PV):
!   m   ::  sample size for 1st group (<= n)
!   n   ::  sample size for 2nd group
!   CV  ::  critical values vector (length 4)
!   PV  ::  probability values vector (length 4)
! DESCRIPTION
! At entry CV_KSJbc(m,n,CV,PV), for m <= n, returns up to four critical
```

```
! values, if available, in vector CV(:), with actual probabilities in PV(:),
! as follows:
!     CV(1) = 1-tailed alpha .05  (2-tailed alpha .10)
!     CV(2) = 1-tailed alpha .025 (2-tailed alpha .05)
!     CV(3) = 1-tailed alpha .01  (2-tailed alpha .02)
!     CV(4) = 1-tailed alpha .005 (2-tailed alpha .01)
!     PV(1) = 1-tailed .05  (2-tailed .10) actual probability
!     PV(2) = 1-tailed .025 (2-tailed .05) actual probability
!     PV(3) = 1-tailed .01  (2-tailed .02) actual probability
!     PV(4) = 1-tailed .005 (2-tailed .01) actual probability
! If a critical value is not available, a -1 is returned instead with p = 0.0
! DECLARE DUMMY VARIABLES
integer, intent(in) :: m, n
integer, intent(out), dimension(:) :: CV
real, intent(out), dimension(:) :: PV
! DECLARE LOCAL VARIABLES
integer, dimension(50,50), save :: CVTbc10, CVTbc05, CVTbc02, CVTbc01
integer :: c, i, ixj, j
real*8, dimension(50,50), save :: PVTbc10, PVTbc05, PVTbc02, PVTbc01
real*8 :: d, pc, prevc
real*8, parameter :: p90=.90, p95=.95, p98=.98, p99=.99
logical :: f10, f05, f02, f01
CVTbc10 = -1   ! initialize CV tables to -1 (indicates no valid entry)
CVTbc05 = -1
CVTbc02 = -1
CVTbc01 = -1
PVTbc10 = 0.0  ! initialize PV tables to zero
PVTbc05 = 0.0
PVTbc02 = 0.0
PVTbc01 = 0.0
! BUILD THE CV AND PV TABLES
Do j = 1,50  ! this is n
  Do i = 1,j  ! this is m
    f10 = .false.
    f05 = .false.
    f02 = .false.
    f01 = .false.
    prevc = 0.0
    ixj = i*j
    Do c = 1,ixj  ! possible critical values
      d = real(c)/real(ixj)  ! Dmn
      pc = akscdf(i,j,d)     ! get the probability of Dmn <= C/(m*n)
      If ((.not.f10).and.(prevc >= p90).and.(pc > prevc)) then
        CVTbc10(i,j) = c
        PVTbc10(i,j) = 1.0 - prevc
        f10 = .true.
      End If
      If ((.not.f05).and.(prevc >= p95).and.(pc > prevc)) then
        CVTbc05(i,j) = c
        PVTbc05(i,j) = 1.0 - prevc
        f05 = .true.
      End If
      If ((.not.f02).and.(prevc >= p98).and.(pc > prevc)) then
        CVTbc02(i,j) = c
        PVTbc02(i,j) = 1.0 - prevc
        f02 = .true.
      End If
```

```
      If ((.not.f01).and.(prevc >= p99).and.(pc > prevc)) then
        CVTbc01(i,j) = c
        PVTbc01(i,j) = 1.0 - prevc
        f01 = .true.
      End If
      prevc = pc
      If ( f10.and.f05.and.f02.and.f01 ) exit
    End Do
  End Do
End Do
return
! ------------------------------------------------------------------------
entry CV_KSJbc(m,n,CV,PV)
CV = -1     ! initialize all return CVs to 'not available'
PV = 0.0    ! initialize all probabilities to zero

If ((1 <= n).and.(n <= 50).and.(1 <= m).and.(m <= n)) then
  CV(1) = CVTbc10(m,n)
  CV(2) = CVTbc05(m,n)
  CV(3) = CVTbc02(m,n)
  CV(4) = CVTbc01(m,n)
  PV(1) = PVTbc10(m,n)
  PV(2) = PVTbc05(m,n)
  PV(3) = PVTbc02(m,n)
  PV(4) = PVTbc01(m,n)
End If
return
! ------------------------------------------------------------------------
end subroutine CV_KSJ_init
! ************************************************************************
real*8 function akscdf(a,b,d)
! From Kim & Jennrich tables of the exact sampling distribution of
! the two-sample Kolmogorov=Smirnov criterion, Dmn, m<=n in
! Selected Tables in Mathematical Statistics, Vol. 1, 77-170
! (1970) Harter & Owens (eds) 2nd printing (1973) with revisions,
! published by American Mathematical Society for the Institute of
! Mathematical Statistics.
! requires a <= b
! DECLARE DUMMY VARIABLES
integer, intent(in) :: a, b
real*8, intent(in) :: d
! DECLARE LOCAL VARIBLES
integer :: i, j
real*8 :: k, w
real*8, allocatable, dimension(:) :: u
allocate(u(b+1))
k = (real(a*b))*d + .5
u(1) = 1.
Do j = 1,b
  u(j+1) = 1.
  If (real(a*j) > k) then
    u(j+1) = 0.
  End If
End Do
Do i = 1,a
  w = real(i)/real(i+b)
  u(1) = w*u(1)
```

```
  If (real(b*i) > k) then
    u(1) = 0.
  End If
  Do j = 1,b
    u(j+1) = u(j) + (u(j+1)*w)
    If (real(IABS(b*i-a*j)) > k) then
      u(j+1) = 0.
    End If
  End Do
End Do
akscdf = u(b+1)
deallocate(u)
return
end function akscdf
! ***********************************************************************
end module CVKSJmod
```

Wilcoxon Rank-sum Test

Main program for printing tables

```
! ***********************************************************************
! program:  CVWRSTJ.exe
! source:   CVWRSTJ.f90
! author:   Bruce R. Fay
! date:     25 Oct 2002  14:22 EDT
! based on: CVWRST.f90 as of 08 Jun 2002 13:02 EDT
! purpose:  Test harness for critical value tables (CVTs) for the
!           Wilcoxon rank sum test for 2-i-s.
! desc:     Provides user choice of critical value module and then
!           outputs results to a file.
! ***********************************************************************
program CVWRSTJ
use CVWRSJ4mod
implicit none
! DECLARE VARIABLES
integer :: i, j, LU, ios, testnum
integer, dimension(:) :: CVi(4)
real*8, dimension(:) :: PVr(4)
! GET USER INPUTS
write(*,*) "Program CVWRSTJ.exe by Bruce R. Fay"
write(*,*)
write(*,*) "Wilcoxon rank-sum test for two independent samples."
write(*,*) "Best-conservative critical values generated by method of"
write(*,*) "Wilcoxon, Katti & Wilcox (1963,68,70,73)."
write(*,*)
write(*,*) "Select one of the following:"
write(*,*)
write(*,*) "0 to exit program or 1 to generate critical values"
write(*,*)
Do
  read(*,*) testnum
  If ((0<=testnum).and.(testnum<=1)) EXIT
  write(*,*) "enter 0 or 1 please"
End Do
```

```
If (testnum==0) GOTO 9999  ! check for user termination
! OPEN FILE FOR OUTPUT AND WRITE HEADER
LU = 8
open(unit=LU, file='CVWRSTJ_.txt', iostat=ios)
IF (ios > 0 ) then
  write(*,*) "Error opening file 'CVWRSTJ_.txt' "
  GOTO 9999
End if
write(LU,*) "File CVWRSTJ_.txt for program CVWRSTJ.exe"
write(LU,*) "by Bruce R. Fay"
write(LU,*)
write(LU,*) "Wilcoxon rank-sum test for two independent samples."
write(LU,*) "Best-conservative critical values generated by method of"
write(LU,*) "Wilcoxon, Katti & Wilcox (1963,68,70,73)."
write(LU,*)
write(LU,*)
! DEFINE FORMATS FOR OUTPUT FILE
100 format(" 1-tailed CVs and PVs at stated nominal alpha levels")
110 format("        -------- nominal alpha -------- &
           &  ----- actual probabilities ----")
120 format(" 1-tail - .05 - - .025  - .01 - - .005  &
           &  - p05 - - p025- - p01 - - p005-")
130 format("  n1 n2")
140 format(1x,2I3,4I8,2x,4F8.4)
! RETRIEVE AND OUTPUT CVs AND PVs
write(*,*) "Generating best-conservative 1-tailed CVs and PVs"
write(*,*) "for WRST for 2-i-s by the method of"
write(*,*) "Wilcoxon, Katti & Wilcox (1963,68,70,73)."
write(*,*)
call CV_WRSJ4_init
write(*,*) "CV_WRSJ4_init completed - CV/PV tables built"
write(LU,100)           ! print header information
write(LU,*)
write(LU,110)           ! print column headers for this format
write(LU,120)
write(LU,130)
write(LU,*)
Do j = 1,50
  Do i = 1,j
    call CV_WRSJ4bc(i,j,CVi,PVr)  ! returned CVs, PVs are 1-tailed
    write(LU,140) i,j,CVi(1:4),PVr(1:4)
  End Do
  write(LU,*)
End Do
! CLOSE FILE
close(unit=LU, status='keep', iostat=ios)
If (ios > 0) then
      write(*,*) "Error closing file 'CVWRSTout_.txt'"
End If
9999 stop
end program CVWRSTJ
```

Module for generating critical values and probabilities

```
! ***************************************************************************
! module:   CVWRS4Jmod
! source:   CVWRS4Jmod.f90
! author:   Bruce R. Fay
! date:     25 Oct 2002 14:04
! based on: CVWRS4mod.f90 as of 08 Jun 2002 12:52 EDT
!           Wilcoxon, Katti & Wilcox (1963) Critical values and
!           probability levels for the Wilcoxon rank sum test (and the
!           Wilcoxon signed rank test), revised Oct 1968, as it appears
!           in Harter & Owen, editors (1970,73) Selected Tables in
!           Mathematical Statistics, Volume I, 171-259.
!           (Values for n1 = 1 and n1 = 2 from Bradley (1968)
!           Distribution-free Statistical Tests, 318, Table III.)
! purpose:  Provide the BEST CONSERVATIVE 1-tailed critical values and
!           associated actual probabilities for the Wilcoxon rank sum
!           test.
! desc:     Generates CV and PV tables on initialization and provides an
!           entry point that returns up to 4 critical values based on the
!           incoming values of n1 and n2.  Checks are made that n1, n2
!           are in the appropriate range and relationship for the tables,
!           with 1 <= n1 <= n2 <=50.
! ***************************************************************************
module CVWRSJ4mod
implicit none
private
public :: CV_WRSJ4_init, CV_WRSJ4bc
contains
! ***************************************************************************
subroutine CV_WRSJ4_init
! INTERFACE
! There are no arguments for CV_WRSJ4_init.  The calling routine must call
! this subroutine once to build the CV table prior to calling CV_WRSJ4bc() to
! obtain critical values for specific m and n.  The calling routine must
! declare two vectors and pass them as arguments: an integer vector of length
! 4 to receive the critical values and a real*8 vector of length 4 to receive
! the associated probabilities.  For entry CV_WRSJ4bc(a,b,CV,PV):
!    a   :: sample size for 1st group (<= b)
!    b   :: sample size for 2nd group
!    CV  :: critical values vector (length 4)
!    PV  :: actual probability values vector (length 4)
! DECLARE DUMMY VARIABLES
integer, intent(in) :: a, b
integer, intent(out), dimension(:) :: CV
real*8, intent(out), dimension(:) :: PV
! DECLARE LOCAL VARIABLES
integer :: h, i, j, k, k1, k2, M, minRS, N, RS, u, ub
integer, dimension(50,50), save :: CVTbc10, CVTbc05, CVTbc02, CVTbc01
real*8, dimension(50,50), save :: PVTbc10, PVTbc05, PVTbc02, PVTbc01
real*8, allocatable, dimension(:) :: cf, f, z
real*8 :: Pr, Prev
real*8, parameter :: p05=0.05, p025=0.025, p01=0.01, p005=0.005
real*8, parameter :: oneppt = 0.001
logical :: f10, f05, f02, f01, Pr_underflow, Prev_underflow
```

```
CVTbc10 = -1  ! initialize CV and PV tables
CVTbc05 = -1
CVTbc02 = -1
CVTbc01 = -1
PVTbc10 = 0.
PVTbc05 = 0.
PVTbc02 = 0.
PVTbc01 = 0.
Do N = 2,50
  Do M = 1,N  ! build the z vector
    minRS = M*(M+1)/2
    k = (M+50)**2
    allocate (z(0:k))
    z = 0.
    Do i = 1,N
      Do j = 1,k
        k1 = (M+i)*j - 1
        K2 = i*j - 1
        If (k1 <= k) then
          z(k1) = z(k1) - real(M+i)
        End If
        If (k2 <= k) then
          z(k2) = z(k2) + real(i)
        End If
        If (k1 > k .and. k2 > k) exit
      End Do
    End Do
!   build the freq and cumfreq vector and find the critical values
    f10 = .false.
    f05 = .false.
    f02 = .false.
    f01 = .false.
    ub = (M+N)*(M+N+1)/2      ! set upper bound on u
    allocate (f(0:ub))        ! allocate the frequency vector
    allocate (cf(0:ub))       !  and the cumulative frequency vector
    f = 0.
    f(0) = 1.
    cf = 0.
    cf(0) = 1.
    Do u = 1,ub
      Do h = 0,(u-1)
        f(u) = f(u) + ( f(h)*z(u-h-1) )
      End Do
      f(u) = f(u)/u
      cf(u) = cf(u-1) + f(u)
      Pr = cf(u)
      Prev = cf(u-1)
      Pr_underflow = .false.
      Prev_underflow = .false.
!     The probabilities Pr and Prev get smaller with each pass
!     through the following loop.  Thus, once they both drop below
!     oneppt (see declaration) there is no point continuing the loop.
      Do i = 1,M
        If (Pr > oneppt) then
          Pr = Pr*(M+1-i)/(N+i)
        Else
          Pr_underflow = .true.
```

```
        End If
        If (Prev > oneppt) then
          Prev = prev*(M+1-i)/(N+i)
        Else
          Prev_underflow = .true.
        End If
        If (Pr_underflow .AND. Prev_underflow) exit
      End Do
      RS = minRS + u-1  ! rank sum = M(M+1)/2 + u-1
!     Find the best conservative CVs for specified alphas
      If ((Prev <= p05).and.(Pr > p05).and.(.not.f10)) then
        CVTbc10(M,N) = RS
        PVTbc10(M,N) = Prev
        f10 = .true.
      End If
      If ((Prev <= p025).and.(Pr > p025).and.(.not.f05)) then
        CVTbc05(M,N) = RS
        PVTbc05(M,N) = Prev
        f05 = .true.
      End If
      If ((Prev <= p01).and.(Pr > p01).and.(.not.f02)) then
        CVTbc02(M,N) = RS
        PVTbc02(M,N) = Prev
        f02 = .true.
      End If
      If ((Prev <= p005).and.(Pr > p005).and.(.not.f01)) then
        CVTbc01(M,N) = RS
        PVTbc01(M,N) = Prev
        f01 = .true.
      End If
      If (f10.and.f05.and.f02.and.f01) exit  ! found all 4 CVs!
    End Do
    deallocate(z,f,cf)
  End Do
End Do
return
! ----------------------------------------------------------------------------
entry CV_WRSJ4bc(a,b,CV,PV)
CV = -1  ! initialize all return CVs to 'not available'
PV = 0.  ! initialize all return p's to zero
If ((b >= 1).and.(b <= 50).and.(a >= 1).and.(a <= b)) then
  CV(1) = CVTbc10(a,b)
  CV(2) = CVTbc05(a,b)
  CV(3) = CVTbc02(a,b)
  CV(4) = CVTbc01(a,b)
  PV(1) = PVTbc10(a,b)
  PV(2) = PVTbc05(a,b)
  PV(3) = PVTbc02(a,b)
  PV(4) = PVTbc01(a,b)
End If
return
! ----------------------------------------------------------------------------
end subroutine CV_WRSJ4_init
! ****************************************************************************
end module CVWRSJ4mod
```