tables of critical values for Kendall's tau

Cory Brunson

Rank correlation takes place at the interface between statistics and combinatorics. Calculating rank correlation coefficients such as Kendall's τ and Spearman's ρ is straightforward and, for small lists, fast; and the distributions for long lists rapidly approach their asymptotics. Kendall's τ mimics Pearson's r by ranging from -1 (perfect negative correlation) through 0 (perfect null correlation) to $+1$ (perfect positive correlation) while remaining nonparametric–unlike ρ, it makes no assumptions about the underlying distribution of values (if such values even exist) that the rankings reflect. Within the τ framework, any of several equivalent statistics may be used to describe the (dis)similarity in two rankings of n objects, and each is easily recovered from the others: P, the number of *concordant* pairs, or pairs of objects ranked the same relative to each other in both lists; Q, the number of *discordant* pairs, so that $P + Q = \binom{n}{2}$; $S = P - Q$; and $\tau = S/\binom{n}{2}$. (Ties are ignored in this discussion.)

There are at least two extant and two apparently orphaned implementations of τ in R: the `kendall` option in the `cor` function (`stats` package) and the eponymous function in the `Kendall` package; and `cor.fk` in the `pcaPP` projection pursuit package and the `kendall` method for the `rpucor` function in the `rpud` NVIDIA package. While it's easy to compute τ values and confidence intervals with these packages, though, i haven't come across a computational resource for critical values, or quartiles, of τ. That is, given n objects and a threshold p, what is the level of concordance (or, equivalently, discordance) between two uniformly random rankings that occurs with probability less than p, and what value $\tau_{n,p}$ corresponds to this concordance? The normal asymptote $\tau_{n,p} \approx \frac{\sqrt{2(2n+5)}}{3\sqrt{n(n-1)}} z_p$ (where $Z \sim N(0,1)$) offers an approximate solution for large values of n, but there will always be a place for exact answers if they're available.

To find a specific critical value using any of these implementations of τ, since these functions do not receive values of τ, P, etc. directly, one must gain control of the number of concordant pairs in the input rankings. This amounts to controlling the number of inversions k in a permutation π of n, taking the identity permutation $(1, 2, \ldots, n)$ as the second ranking:

```
# Count inversions in a vector
vec.inv <- function(vec) {
    sum(unlist(sapply(1:(length(vec) - 1), function(i) {
        sapply((i + 1):length(vec), function(j) vec[i] > vec[j])
    })))
}
```

Whereas the identity has zero inversions, k here plays the role of Q. (And, since τ is symmetric, it's enough to consider k values from 0 to $\lceil \binom{n}{2}/2 \rceil$.) To find such a permutation, recall (or find out for yourself, or just take my word for it) that the number of inversions of π equals the minimal length of a word for π, and one minimal word for the longest permutation $\pi_0 = (n, n-1, \ldots, 1)$ is

$$\prod_{j=1}^{n-1} \left(\prod_{i=j}^{1} s_i \right) = (s_1)(s_2 s_1) \cdots (s_{n-1} s_{n-2} \cdots s_1).$$

Therefore, a (highly non-arbitrary) permutation of length k obtains as the product of the first (rightmost) k generators in this word. Here is an implementation in R (with an option to leverage the symmetry for a bit of efficiency):

```
# Generate a permutation of 1:n having a given number k of inversions
inv.vec <- function(n, k, bw.ok = TRUE) {
    if(k > choose(n, 2) | k < 0 | any(c(n, k) %% 1 != 0)) stop('invalid n or k')
```

```
    backwards <- (k > choose(n, 2) / 2) & bw.ok
    if(backwards) k <- choose(n, 2) - k
    vec <- 1:n; ninv <- 0
    while(ninv < k) {
        i <- vec[1]
        j <- if(k - ninv < n - i) (2 + k - ninv):n else {
            if(i == 1) c() else (2 + n - i):n
        }
        vec <- c(vec[-c(1, j)], i, vec[j])
        ninv <- ninv + {
            if(k - ninv < n - i) (k - ninv) else (n - i)
        }
        stopifnot(ninv == vec.inv(vec))
    }
    if(backwards) vec <- rev(vec)
    return(vec)
}
```

From there it is quick work to whittle down the options. For example, if i want to know the "value to beat" for a type-I error $p < .01$ on a one-sided test for positive correlation between two rankings of $n = 30$ objects using the `Kendall` package, i can run the following:

```
n <- 30; p <- .01
# range of revelant k (at most half of all possible discordant pairs)
k.ran <- c(0, floor(choose(n, 2) / 2))
# corresponding one-sided p-values
p.ran <- sapply(k.ran, function(k) Kendall(x = 1:n, y = inv.vec(n, k))$sl / 2)
# interpolate (linearly)
repeat {
    if(p <= p.ran[1]) {
        k <- NA; break
    }
    if(p > p.ran[2]) {
        k <- k.ran[2]; break
    }
    q <- (p - p.ran[1]) / diff(p.ran)
    k <- round(q * diff(k.ran) + k.ran[1])
    if(k == k.ran[1]) k <- k + 1 else if(k == k.ran[2]) k <- k - 1
    k.p <- Kendall(x = 1:n, y = inv.vec(n, k))$sl / 2
    if(k.p == p) break else if(k.p < p) {
        k.ran[1] <- k; p.ran[1] <- k.p
    } else if(k.p > p) {
        k.ran[2] <- k; p.ran[2] <- k.p
    }
    if(diff(k.ran) < 2) {
        k <- k.ran[1]; break
    }
}
tau.K  <- round((choose(n, 2) - 2 * k) / choose(n, 2), 4)
print(paste('need tau >', tau.K))
```

```
## [1] "need tau > 0.3057"
```

This result disagrees slightly with the value 0.301 in Peter M Lee's critical values table, possibly due to error in `Kendall`, which cites this algorithm by Best and Gipps. The same procedure with `Kendall(...)$sl` exchanged for `cor.test(..., alternative = 't', method = 'kendall')$p.value`, which is exact for small n and computed for the asymptotic Z-score otherwise, yields a critical value of 0.3011.

All this, however, makes for a very inefficient means to generate the titular table of critical values of τ. For that, rather than constructing a permutation of a desired length having a desired number of inversions, then iterating this construction over many lengths and inversion counts, it makes sense, provided it is possible, both

a. to produce the distribution of the inversion counts $(0, 1, \ldots, \binom{n}{2})$ for each length n, and
b. to compute critical values along the (very probably) recursive process that yields these distributions, rather than producing each distribution from scratch.

Conveniently, Shashank at StackOverflow has already made the key observation that these distributions are the Mahonian numbers $T(n, k)$, whose generating functions take the elegant form:

$$\prod_{j=0}^{n-1} \left(\sum_{i=0}^{j} x^i \right) = \sum_{k=0}^{\binom{n}{2}} T_{n,k} x^k,$$

which implies that they satisfy a simple recursive formula:

```
# Compute the Mahonian numbers
mahonians.recursive <- function(n) {
    if(n <= 0 | n%%1 != 0) stop('n must be a positive integer')
    if(n == 1) 1 else rowSums(sapply(0:(n - 1), function(i) {
        c(rep(0, i), mahonians.recursive(n - 1), rep(0, n - 1 - i))
    }))
}
print(paste('Example: the Mahonian numbers for n = 5 are',
            toString(mahonians.recursive(5))))
```

```
## [1] "Example: the Mahonian numbers for n = 5 are 1, 4, 9, 15, 20, 22, 20, 15, 9, 4, 1"
```

This formula can now serve as a basis for the construction of a critical values table for τ with whatever maximum length n and suite of p-value thresholds one desires; though the external recursion slows it down dramatically, so for the implementation an internal loop is used. (For convenience, the code below produces a table of critical values of k, from which those of τ can be recovered by arithmetic.)

```
# Rapid table of critical values (n a vector, alpha a vector)
tau.crit.table <- function (n, alpha, incl.len = TRUE, stat = "tau") {
    if (any(n <= 0 | n%%1 != 0))
        stop("n must be a positive integer")
    if (any(alpha > 0.5 | alpha <= 0))
        stop("alpha must be positive and below .5")
    vec <- 1
    m <- 1
    max.n <- max(n)
    mat <- matrix(NA, nr = if (min(n) == 1)
        1
    else 0, nc = length(alpha))
    while (max.n > m) {
```

```
        vec <- rowSums(sapply(0:m, function(i) {
            c(rep(0, i), vec, rep(0, m - i))
        }))
        m <- m + 1
        if (m %in% n) {
            s <- cumsum(vec[1:floor(choose(m, 2)/2)])/sum(vec)
            i <- sapply(alpha, function(a) if (s[1] < a)
                max(which(s < a)) - 1
            else NA)
            mat <- rbind(mat, i)
        }
    }
    stopifnot(length(vec) == choose(max.n, 2) + 1)
    colnames(mat) <- alpha
    rownames(mat) <- n
    if (stat == "tau") {
        mat <- (choose(n, 2) - 2 * mat)/choose(n, 2)
    }
    else if (stat == "P" | grepl("^concord", stat, ignore.case = TRUE)) {
        mat <- choose(n, 2) - mat
    }
    else if (stat == "S") {
        mat <- choose(n, 2) - 2 * mat
    }
    else if (!(stat %in% c("K", "k", "Q")) & !grepl("^discord|^inv",
        stat, ignore.case = TRUE)) {
        stop("unknown statistic")
    }
    if (incl.len)
        mat <- cbind(n = n, mat)
    return(mat)
}
```

As an example, here are the critical values (of k and of τ) for lengths at intervals of 10 up to 100 and type-I error thresholds at negative powers of 10:

```
# Ranges of n and of alpha
example.n <- seq(10, 100, 10); example.alpha <- 10 ^ (-1:-4)
# Critical value table for k
inv.crit.vals <- tau.crit.table(n = example.n,
                                alpha = example.alpha,
                                incl.len = TRUE, stat = 'inv')
print(inv.crit.vals)
```

```
##       n  0.1 0.01 0.001 1e-04
## 10   10   14    9     5     3
## 20   20   74   59    48    40
## 30   30  180  152   132   116
## 40   40  334  290   258   233
## 50   50  535  473   429   394
## 60   60  783  702   644   597
## 70   70 1080  978   904   845
## 80   80 1425 1300  1210  1137
```

```
## 90   90 1817 1669  1561  1474
## 100 100 2259 2084  1958  1856
```

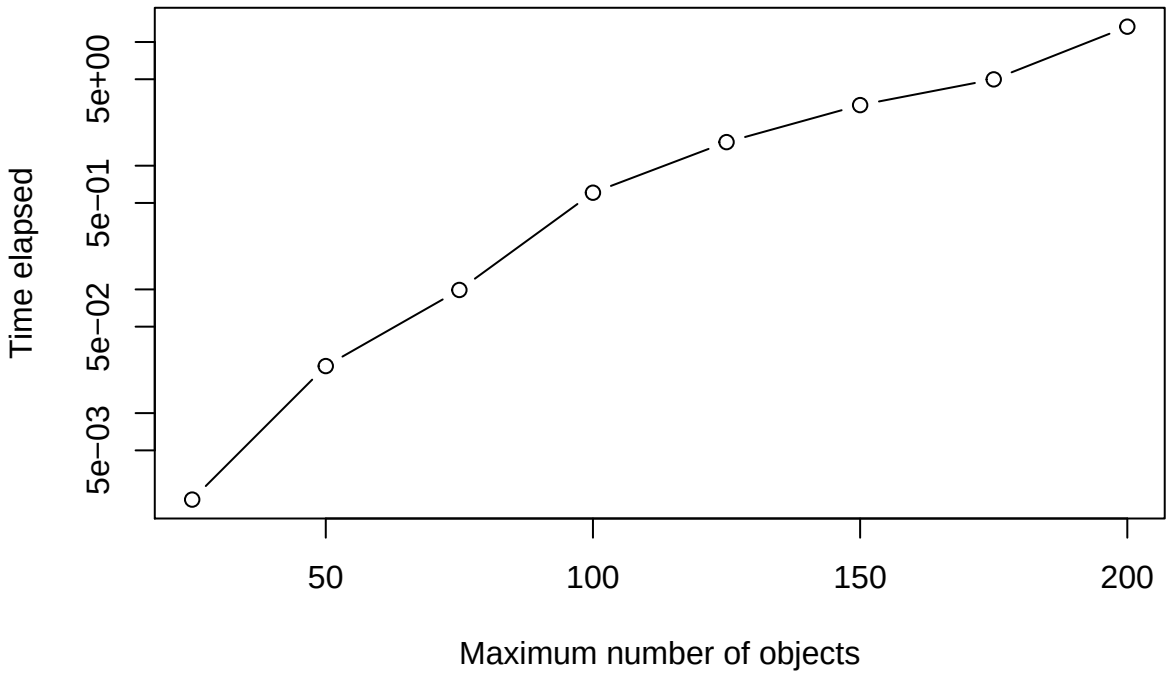
```
# Critical value table for tau
tau.crit.vals <- tau.crit.table(n = example.n,
                                alpha = example.alpha,
                                incl.len = TRUE)
print(tau.crit.vals)
```

```
##       n        0.1      0.01     0.001     1e-04
## 10   10 0.37777778 0.6000000 0.7777778 0.8666667
## 20   20 0.22105263 0.3789474 0.4947368 0.5789474
## 30   30 0.17241379 0.3011494 0.3931034 0.4666667
## 40   40 0.14358974 0.2564103 0.3384615 0.4025641
## 50   50 0.12653061 0.2277551 0.2995918 0.3567347
## 60   60 0.11525424 0.2067797 0.2723164 0.3254237
## 70   70 0.10559006 0.1900621 0.2513458 0.3002070
## 80   80 0.09810127 0.1772152 0.2341772 0.2803797
## 90   90 0.09263421 0.1665418 0.2204744 0.2639201
## 100 100 0.08727273 0.1579798 0.2088889 0.2501010
```

The agreement of several rows with their analogs in Lee's table validates the procedure; other tables exist for others of the equivalent statistics i mentioned above, e.g. the number of concordant pairs and the numerator S, which the interested reader can check for themselves. Most of these functions, and a few more, are available in package form, in accordance with the Leek Group's advice.

As a final test, here are the runtimes for tables up to maximum n values of 25 through 200 at intervals of 25, performed on a 3.2 GHz Intel Core i5, which hint at the slow combinatorial explosion that eventually undermines the usefulness of the method, though far beyond the point at which the normal approximaiton becomes adequate:

```
N <- seq(25, 200, 25)
st.mat <- sapply(N, function(n) {
    system.time(tau.crit.table(n = n,
                               alpha = 10^(-1:-3),
                               incl.len = TRUE))
})
plot(x = N, y = st.mat[3, ], type = 'b', log = 'y',
     xlab = 'Maximum number of objects', ylab = 'Time elapsed')
```


Time elapsed
5e−03
5e−02
5e−01
5e+00
50
100
150
200
Maximum number of objects