Teresa Ledwina, Grzegorz Wyłupek
Institute of Mathematics
Polish Academy of Sciences
Kopernika 18, 51-617 Wrocław, Poland

R codes to: Nonparametric tests for stochastic ordering. TEST (2011). DOI 10.1007/s11749-011-0278-7

Description

Here we provide the computer programs which let one compute the values of the statistics M_d, Q_T and Q_S, the standardized empirical Fourier coefficients L_j's as well as the values of the rules T and S.

The notations in the program are consistent with or similar to those used in the paper. Let $\mathbf{x} = (x_1, \ldots, x_m)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ denote the two samples of sizes m and n, respectively, and set $N = m + n$. In the program we rename the functions l_j and L_j to `l.j` and `L.j`. Similarly, `k.N` and `d.N` stand there for the quantities $k(N)$ and $d(N)$. The choice of `k.N` and thereby `d.N` belongs to the user and should reflect the individual aims and needs. Some authors' recommendation is given in Section 5.2.

The code starts with the definition of the functions `l.j` and `L.j`. The procedure `test.M.d` returns the values of the statistic M_d and the vector of standardized empirical Fourier coefficients $L_j = \sqrt{mn/N}\,\hat{\gamma}_j$, while, given the tuning parameter t, the routine `test.Q` yields the values of the data-driven statistics Q_T and Q_S as well as the dimensions selected by the rules T and S.

Calculation of M_d, Q_T and other quantities

```
l.j = function(k,i,z){
  a.j = (2*i-1)/2^(k+1)
  left = -sqrt((1-a.j)/a.j)
  right = sqrt(a.j/(1-a.j))
  score = ifelse(z < a.j, left, right)
  return(score)
}

L.j = function(k,i,x,y,m,n){
  N = m + n
  H = ecdf(c(x,y))
  rx = H(x) - 1/(2*N)
  ry = H(y) - 1/(2*N)
  cx = sum( l.j(k,i,rx) )/ m
  cy = sum( l.j(k,i,ry) )/ n
  score = (cy - cx)*sqrt(m*n/N)
  return(score)
}
```

```
test.M.d = function(x,y,m,n,k.N){
  N = m+n
  d.N = 2^(k.N+1)-1
  L.vec = matrix(0,1,d.N)
  for(k in 0:k.N){
    for(i in 1:(2^k)){
      j = 2^k - 1 + i
      L.vec[1,j] = L.j(k,i,x,y,m,n)
    }
  }
  M.d = min( L.vec[1,] )
  result = c(M.d,L.vec)
  return(result)
}

test.Q = function(x,y,m,n,k.N,t){
  N = m+n
  d.N = 2^(k.N+1)-1
  L.vec = matrix(0,1,d.N)
  L.vec.trun = matrix(0,1,d.N)
  for(k in 0:k.N){
    for(i in 1:(2^k)){
      j = 2^k - 1 + i
      L.vec[1,j] = L.j(k,i,x,y,m,n)
      L.vec.trun[1,j] = max( - L.vec[1,j], 0 )
    }
  }
  Q.d.vec = matrix(0,1,k.N+1)
  D.vec = 2^(0:k.N+1) - 1
  for(k in 1:(k.N+1)){
    Q.d.vec[1,k] = L.vec.trun[1,1:D.vec[k]] %*% L.vec.trun[1,1:D.vec[k]]
  }
  S = which.max( Q.d.vec[1,] - D.vec*log(N) )
  M = which.max( Q.d.vec[1,] )
  if( max( -L.vec[1,] ) <= sqrt(t*log(N)) ){
    T = S
  }else{
    T = M
  }
  Q.S = Q.d.vec[1,S]
  Q.T = Q.d.vec[1,T]
  result = c(Q.T,Q.S,T,S)
  return(result)
}
```