

1 Introduction

The C functions for computing the test statistics used in the article "Bootstrap Tests for Multivariate Directional Alternatives" are provided here. These functions utilizes the IMSL routines for computing the projections described in the articles. This is reflected in the initial include statements in the programs.

Both the test statistics U for BT test and D for MBT test are given. These functions are to be compiled in conjunction with interface programs for bootstrapping in C or Splus.

Further inquiries about these should be directed to Abu Minhajuddin, Division of Biostatistics, Department of Clinical Sciences, UT Southwestern Medical Center, Dallas, Texas.

My email address is abu.minhajuddin@utsouthwestern.edu.

```
#include<stdio.h>
#include<math.h>
#include<imsls.h>
#include<imsl.h>
#include<stdlib.h>

/*****
/*Function invert inverts a matrix of order p      p dim */
*****/
void invert(double *Mat, long *p){
    long i, j ,k;
    double c, EPS=0.000001;
    for(k=0;k<*p;k++){
        if (ABS(c=Mat[k+(*p)*k]) < EPS){
            return;
        }
        for(i=0;i<*p;i++){
            for(j=0;j<*p;j++){
                if(i != k && j != k){
                    Mat[i+(*p)*j] -= (Mat[i+(*p)*k]*Mat[k+(*p)*j])/c;
                }
            }
        }
        for(i=0;i<*p;i++){
            if(i != k ){
                Mat[i+(*p)*k] /= -c;
                Mat[k+(*p)*i] /= c;
            }
        }
        Mat[k+(*p)*k] = 1.0/c;
    }
    return;
}
```

```

}

/*****
/*   This is an utility function for the function Projection   */
/*   Generates possible combinations.....needed for function proj */
*****/
void binari(long *bin, long p){
    long i, j, m;
    m = pow(2,p);
    for(i=0; i<m;i++){
        if(i == 0){
            for(j=0;j<p;j++){
                bin[i+m*j] = 0;
            }
        }
        else{
            for(j=0;j<p;j++){
                bin[i+m*j] = bin[(i-1)+m*j];
            }
            j=0;
            while(bin[i+m*j] == 1){
                bin[i+m*j] = 0;
                j++;
            }
            bin[i+m*j] = 1;
        }
    }
}

/*****
/*   This is an utility function for function projection.   */
/*   Actually computes projection in p dim, not necessarily */
/*   the closest one                                         */
*****/
void proj(double *A, double *x, double *ax, long p, long *ind, double *m){
    double *CC, *CX, *mp;
    long i, j, k, len=0, rid=0, cid=0;
    for(i=0; i<p;i++){
        m[i] = 0.0;
        len +=ind[i];
    }

    CC = (double *) calloc(len*len, sizeof(double));
    CX = (double *) calloc(len, sizeof(double));
    mp = (double *) calloc(len, sizeof(double));

```

```

    for(i=0;i<p;i++){
        if(ind[i] > 0){
            CX[rid] = ax[i];
            rid +=1;
        }
    }
    rid = 0;
    for(i=0;i<p;i++){
        if(ind[i] > 0){
            for(j=0;j<p;j++){
                if(ind[j] >0){
                    CC[rid + len*cid] = A[i+p*j];
                    cid +=1;
                }
            }
            rid +=1; cid=0;
        }
    }
    invert(CC,&len);
    for(i=0;i<len;i++){
        mp[i] = 0.0;
        for(j=0;j<len;j++){
            mp[i] += CC[i+j*len]*CX[j];
        }
    }
    k=0;
    for(i=0;i<p;i++){
        if(ind[i] >0){
            m[i] = mp[k];
            k++;
        }
    }
    free(CC); free(CX); free(mp);
}

/*****
/* Projection of a vector x onto the boundary of the positive orthant in p dim */
/* When is x is inside the positive orthant */
/* */
/* Input: */
/* A: Square matrix of order p that contains (n-1)*S, */
/* where S is the sample covariance matrix */
/* avgx: Vector of sample mean times sqrt(n) */
/* n: sample size */
/* p: dimension of the sample */
/* */

```

```

/* Output: */
/* mp: projection of avgx on to the boundary of positive orthant */
/*****/
void projection(double *A, double *x, long p, double *m){
    double *cx, *mx, *diff, q, normc=0.0, cnt = 99999.0;
    long i, j, k, l, *ind, *bin, mm, sum=0, id;
    mm = pow(2, p);

    ind = (long *) calloc(p, sizeof(long));
    cx = (double *) calloc(p, sizeof(double));
    mx = (double *) calloc(p, sizeof(double));
    diff = (double *) calloc(p, sizeof(double));
    bin = (long *) calloc(mm*p, sizeof(long));

/* Making CX */
    for(i=0;i<p;i++){
        for(j=0;j<p;j++){
            cx[i] +=A[i+p*j]*x[j];
        }
    }

/* Making ind */
    binari(bin, p);

/* Checking each case */
    for(i=0;i<mm;i++){
        sum = 0;
        for(j=0;j<p;j++){
            sum +=bin[i+mm*j];
        }
        if(sum > 0 && sum < p){
            for(j=0;j<p;j++){
                ind[j] = bin[i+mm*j];
            }
            proj(A, x, cx, p, ind, mx);
            id = 0;
            for(j=0;j<p;j++){
                if(mx[j] < 0.0){
                    id = 1;
                }
            }
            if(id != 1){
                for(j=0;j<p;j++){
                    diff[j] = x[j] - mx[j];
                }
            }
            normc = 0.0;
        }
    }
}

```

```

    mnorm(&p, A, diff, &normc);
    if(normc < cnt){
        cnt = normc;
        for(j=0;j<p;j++){
            m[j] = mx[j];
        }
    }
}
}
}
}
free(ind); free(cx); free(mx); free(diff); free(bin);
}

/*****
/*  Function sample does the bootstrap resampling p dim      */
/*****
void sample(double *x, long *n, long *p, double *y, double *seed){
    long i, l;
    int *id, ss;
    ss = (*seed);
    imsls_random_seed_set(ss);
    id = imsls_d_random_uniform_discrete(*n, *n, 0);
    *seed = imsls_random_seed_get();
    for(i=0;i<*n;i++){
        for(l=0;l<*p;l++){
            y[i + (*n)*l] = x[id[i]-1 + (*n)*l];
        }
    }
    free(id);
    return;
}

/*****
/*      Function mnorm computes the norm of x wrt A    (p dimension)  */
/*  Input:                                             */
/*  p: Length of vector x                                     */
/*  A: Matrix of order p-square (matrix for computing norm) */
/*  x: p-vector, usually the vector of average here        */
/*                                                         */
/*  Output:                                                 */
/*  norm: norma of x wrt the matrix A                      */
/*****
double mnorm(long *p, double *A, double *x, double *norm){
    long i, j;
    *norm=0.0;
    for(i=0;i<*p;i++){

```

```

        for(j=0;j<*p;j++){
            *norm +=x[i]*x[j]*A[i+(*p)+j];
        }
    }
    return(*norm);
}

/*****
/* Function findmp computes the projection of a vector x
/* onto positive orthant (p dimension) when x is
/* not in the positive orthant.
/*
/* Input:
/* A: Square matrix of order p that contains (n-1)*S,
/* where S is the sample covariance matrix
/* avgx: Vector of sample mean times sqrt(n)
/* n: sample size
/* p: dimension of the sample
/*
/* Output:
/* mp: projection of avgx on to the boundary of positive orthant
*****/
void findmp(double *A, double *avgx, long *p, double *mp){
    double *bf, *gf, *hf, *mmf, eps=0.000000000000001;
    long i, j, m, n, meq=0, idd=0;
    m = n = *p;
    bf = (double *) calloc((int)*p, sizeof(double));
    gf = (double *) calloc((int)*p, sizeof(double));
    hf = (double *) calloc((int)(*p)*(*p), sizeof(double));
    for(i=0;i<*p;i++){
        if(avgx[i]<0.0){
            idd = 1;
        }
    }
    if (idd > 0){
        for(i=0;i<*p;i++){
            gf[i] = 0.0; bf[i] = 0.0;
            for(j=0;j<*p;j++){
                bf[i] += A[i+(*p)*j]*avgx[j];
                if(i != j){
                    hf[i+(*p)*j] = A[i+(*p)*j] + A[j+(*p)*i];
                }
                else{
                    hf[i+(*p)*j] = 2.0*A[i+(*p)*j];
                }
            }
        }
    }
}

```

```

    }
    mmf = imsl_d_quadratic_prog(m,n,meq,A,bf,gf,hf,0);
    for(i=0;i<*p;i++){
        if(mmf[i]>(-1.0)*eps && mmf[i] < eps){
            mmf[i] = 0.0;
        }
        mp[i] = mmf[i];
    }
}
else{
    for(i=0;i<*p;i++){
        mp[i] = avgx[i];
    }
}
free(bf); free(hf); free(gf);
}

/*****
/* Function U computes the the statistic U for the test BT (p dimension) */
/*
/* Input:
/*   A: Square matrix of order p that contains (n-1)*S,
/*       where S is the sample covariance matrix
/*   mx: Vector of sample mean times sqrt(n)
/*   n: sample size
/*   p: dimension of the sample
/*
/* Output:
/*   Uc: Test statistic U for BT
*****/
void U(double *A, double *mx, long *n, long *p, double *Uc){
    double *m, normc = 0.0, *mmx,eps=0.0000000001;
    long i,ind=0;
    m = (double *) calloc(*p, sizeof(double));
    mmx = (double *) calloc(*p, sizeof(double));
    for(i=0;i<*p;i++){
        m[i] = 0.0;
    }
    findmp( A, mx, p, m);
    mnorm(p, A, m, &normc);
    *Uc = normc;
    for(i=0;i<*p;i++){
        mmx[i] = mx[i] - m[i];
    }
    normc = 0.0;
    mnorm(p, A, mmx, &normc);
}

```

```

    *Uc = (*Uc)/(1.0+normc);
    free(m); free(mmx);
}

/*****
/* Function D computes the the statistic U for the test MBT (p dimensions) */
/*
/* Input:
/*   A: Square matrix of order p that contains (n-1)*S,
/*       where S is the sample covariance matrix
/*   mx: Vector of sample mean times sqrt(n)
/*   n: sample size
/*   p: dimension of the sample
/*
/* Output:
/*   Uc: Test statistic U for MBT
*****/
void D(double *A, double *mx, long *n, long *p, double *Dc){
    double *m, *md, normc=0.0,eps=0.0000000001;
    long i,j,idd=0;

    m = (double *) calloc(*p, sizeof(double));
    md = (double *) calloc(*p, sizeof(double));
    for(i=0;i<*p;i++){
        m[i] =0.0; md[i] = 0.0;
        if(mx[i]<0.0){
            idd = 1;
        }
    }
    if(idd > 00){
        *Dc = 0.0;
    }
    else{
        for(i=0;i<*p;i++){
            m[i] = 0.0;
        }
        projection(A, mx, *p, m);
        for(i=0;i<*p;i++){
            if(m[i] > -1.0*eps && m[i] < eps){
                m[i] = 0.0;
            }
        }
        for(i=0;i<*p;i++){
            md[i] = mx[i] - m[i];
        }
        mnorm(p, A, md, &normc);
    }
}

```

```
        *Dc = normc;  
    }  
    free(m); free(md);  
}
```