

FILE S2

Implementation of the weight calculation in ANSI C

```

/* LAPACK routines needed */

#define sqr(x) ((x)*(x))

/* function declaration */

void show(double *X,int m,int n);

int weights(double *wmean, double *wvar, int nind, double *D, double thresh);

void savesparse(FILE *fp1, FILE *fp2,double *wmean,int nind);

void mconv(float *D,double *D1,int nind);

void gmatrix(FILE *fpG, double *G, int nind);

void get(double *G,double *G12, double *G11, double *G22, int npar, int it, int nind);

void minvert(double *D,int nind);

void savevar(double *Gtmp, double *G22, int npar, FILE *fp4);

void saveexp(double *G1211, double *wmean, int npar, int l, int nind);

void get22(double *G22, double *G, int npar, int nind);

int main(int argc, char **argv)
{

int i,npar,nn,nblock,nnb,l,nind;
FILE *fp1,*fp2,*fp3,*fp4,*fp5; /* file pointers */
double *G,*Gtmp,*G12,*G11,*G1211,*wmean,*G22;
float thresh;
char *inp,gfile[50];
printf("\n");
printf("***** starting main program *****\n\n");
/****** read data from input file *****/

if(argc > 1)
inp = argv[1];

```

```

else

inp = "input.txt";

printf("Name of input file: %s\n",inp);

if(!(fp1 = fopen(inp, "r"))){
printf("Error, could not open user defined data file!\n");
return 0;
}

fscanf(fp1, "%*[\n]"); /* skip header in file input.txt */
fscanf(fp1, "%s%*[\n] %d%*[\n] %f%*[\n] %d%*[\n]",gfile,&nind,&thresh,&npar);
fclose(fp1);
printf("No of individuals in pedigree: %d\n",nind);

/* open data files */

if(!(fp2 = fopen(gfile, "r")) ||
!(fp3 = fopen("nind", "w")) ||
!(fp4 = fopen("OutVar", "w")) ||
!(fp5 = fopen("ntot", "w"))){
printf("Error, could not open data file!\n");
return 0;
}

/* defining help variables */
nn = sqr(nind);
n nb = sqr(npar);

/* check that nblock is a integer! */
if(nind%npar!=0){
printf("Total number of blocks not an integer. Please check your input data\n");

```

```

    return 0;
}
nblock = nind/npar;
printf("No of partitions of covariance structure: %d\n",nblock);

/***** dynamic memory allocation *****/

if(!(G = (double *) malloc(nn*sizeof(double))) ||
    !(wmean = (double *) malloc(nn*sizeof(double))) ||
    !(G22 = (double *) malloc(nnb*sizeof(double))) ||
    !(Gtmp = (double *) calloc(nnb,sizeof(double)))){ /* npar x npar */
printf("Error in allocating memory\n\n");
return 0;
}

/* read relationship matrix from file */
gmatrix(fp2, G, nind);
fclose(fp2);

get22(G22,G,npar,nind); /* compute weight for first block of parameters */

savevar(Gtmp, G22, npar, fp4);

for(i = 1; i < nblock; i++){ /* looping over all parameters */
printf("i = %d\n",i);

l = i*npar; /* help variable */

if(!(G12 = (double *) malloc(l*npar*sizeof(double))) || /* npar x l */
    !(G1211 = (double *) malloc(l*npar*sizeof(double))) || /* npar x l */
    !(G11 = (double *) malloc(sqr(l)*sizeof(double)))){ /* l x l */

```

```
printf("Error in allocating memory\n\n");
return 0;
}

/* compute expectation vector */
/* pick out conditioned part of covariance matrix */
get(G,G12,G11,G22,npar,i,nind);

/* inverting conditioned part of covariance matrix */
minvert(G11,l);

dsymm('R', 'L', npar, l, 1., G11, l, G12, npar, 0., G1211, npar);
/* compute covariance matrix */
dgemm('N', 'T', npar, npar, l, 1., G1211, npar, G12, npar, 0., Gtmp, npar);

/* save parameters */
saveexp(G1211, wmean, npar, l, nind);
savevar(Gtmp, G22, npar, fp4);

free(G12);
free(G11);
free(G1211);
}

savesparse(fp5, fp3, wmean, nind); /* save non-zero weights for mean calculation */

free(G22);
free(G);
free(Gtmp);
free(wmean);
```

```
fclose(fp3);
fclose(fp4);
fclose(fp5);

printf("***** leaving main program *****\n\n");

return 1;
}
void saveexp(double *G1211, double *wmean, int npar, int l, int nind){

int i,j,l1=l+npar,i1;

for(i = l; i < l1; i++){
    i1 = i-l;
    for(j = 0; j < l; j++){
        wmean[j*nind+i] = G1211[j*npar+i1];
    }
}

void savevar(double *Gtmp, double *G22, int npar, FILE *fp4){

int l,j;

for(j = 0; j < npar; j++){
    for(l = 0; l < npar; l++){
        fprintf(fp4,"%f\t",G22[l*npar+j]-Gtmp[l*npar+j]);
        fprintf(fp4,"\n");
    }
}

void get(double *G,double *G12, double *G11, double *G22, int npar, int it, int nind){
```

```
int i,j,n=it*npar,n1=(it+1)*npar,l,k;
```

```
for(i = 0; i < n; i++)
```

```
for(j = 0; j < n; j++)
```

```
    G11[j*n+i] = G[j*nind+i];
```

```
for(i = n; i < n1; i++){
```

```
    l=i-n;
```

```
    for(j = 0; j < n; j++)
```

```
        G12[j*npar+l] = G[j*nind+i];
```

```
}
```

```
for(i = 0; i < npar; i++){
```

```
    l = i+n;
```

```
    for(j = 0; j < npar; j++){
```

```
        k = j+n;
```

```
        G22[j*npar+i] = G[k*nind+l];
```

```
    }
```

```
}
```

```
}
```

```
void get22(double *G22, double *G, int npar, int nind){
```

```
int i,j;
```

```
for(i = 0; i < npar; i++)
```

```
for(j = 0; j < npar; j++)
```

```
    G22[j*npar+i] = G[j*nind+i];
```

```
}
```

```
void show(double *X,int m,int n){

    int i,j;

    for (i = 0; i < m; i++){
        for (j = 0; j < n; j++){
            printf(" %1.3f ",X[j*m+i]);
            printf("\n");
        }
    }

}

int weights(double *wmean, double *wvar, int nind, double *D, double thresh){

    printf("Entering weights function.....\n");

    int i,j,k,m,info, *ipiv;
    double *Dtmp,*dtmp,*dtmp2,tmp; //,*dtot
    char transa = 'N';
    int hit1=0,hit2=0;

    for(i = 1; i <= nind; i++){ /* skip first element */
        printf("i = %d\n",i);
        if(i > 1){
            m = i-1;

            Dtmp = (double *)malloc(sqr(m)*sizeof(double));
            dtmp = (double *)malloc(m*sizeof(double));
            dtmp2 = (double *)malloc(m*sizeof(double));
            ipiv = (int *)malloc(m*sizeof(int));
```

```
for(j = 0; j < m; j++){
    for(k = 0; k < m; k++){
        Dtmp[k*m+j] = D[k*nind+j];
        dtmp[j] = D[j*nind+m];
        dtmp2[j] = D[j*nind+m];
    }

/* LU factorization of real m by n matrix
*/
dgetrf(m, m, Dtmp, m, ipiv, &info);
if(info!=0){
    printf("Error when calling function dgetrf\ninfo = %d\n",info);
    return 0;
}

dgetrs(transa, m, 1, Dtmp, m, ipiv, dtmp, m, &info);
if(info!=0){
    printf("Error when calling function dgetrs\ninfo = %d\n",info);
    return 0;
}

tmp = 0.;

/* store parameters in wmean, wtmp */

for(j = 0; j < m; j++){
    if(fabs(dtmp[j]) > thresh)
        wmean[j*nind+m] = dtmp[j];
    tmp += dtmp[j]*dtmp2[j];
}
}
```



```
    free(dtmp);
    free(dtmp2);
    //free(dtot);
    free(ipiv);
    wvar[i-1] = 1.-tmp;
    if(wvar[i-1] < 0.){
        wvar[i-1] = 0.00001;
        hit1++;
    }
    if(wvar[i-1] > 1.){
        wvar[i-1] = 1.;
        hit2++;
    }
    printf("wvar[%d] = %f\n",i-1,wvar[i-1]);
    free(Dtmp);

}
}

wvar[0] = 1.;
return 1;

}
void gmatrix(FILE *fpG, double *G, int nind){

    int i,j;
    float *Gtmp;

    Gtmp = (float *)malloc(nind*nind*sizeof(float));
```

```

for (i = 0; i < nind; i++)
  for (j = 0; j < nind; j++){
    fscanf(fpG,"%f\t", &Gtmp[j*nind+i]);
    G[j*nind+i]=(double)Gtmp[j*nind+i];
  }

free(Gtmp);

}

void savesparse(FILE *fp1, FILE *fp2,double *wmean,int nind){

// compressed sparse format
int i,j,n=0,l=0,hit,n0=0,ntot=0,k=0;
int *ja,*ia,*nid,*nel,*nst;
double *ra;
double *na;

// computing the number of non-zero elements
for(i = 0; i < nind; i++){
  n0 = n;
  for(j = 0; j < nind; j++){
    if(wmean[j*nind+i] > 0.00000001 || wmean[j*nind+i] < -0.00000001){
      n++;
    }
  }
}

if(n == n0)
  ntot ++;
}

ntot += n;

printf("length of na: %d\n",ntot);

```

```
printf("Number of non-zero elements: %d\n",n);
ra = (double *)malloc(n*sizeof(double));
nid = (int *)calloc(ntot,sizeof(int));
nel = (int *)malloc(nind*sizeof(int));
nst = (int *)malloc(nind*sizeof(int));
na = (double *)calloc(ntot,sizeof(double));
ja = (int *)malloc(n*sizeof(int));
ia = (int *)calloc((nind+1),sizeof(int));

for(i = 0; i < nind; i++){
    hit = 0;
    n0 = 1;
    for(j = 0; j < nind; j++){
        if(wmean[j*nind+i] > 0.00000001 || wmean[j*nind+i] < -0.00000001){
            ra[l] = wmean[j*nind+i];
            na[k] = ra[l];
            nid[k] = j+1;
            ja[l] = j+1;
            l++;
            k++;
            if(hit == 0){
                ia[i] = 1;
                nst[i] = k;
                hit = 1;
            }
        }
    }
}

if(l == n0){
    k++;
    nst[i] = k;
}
```

```
    nel[i] = l-n0;
}
ia[nind] = n+1;

for(i = 0; i < nind; i++){
    if(ia[i] == 0)
        ia[i] = 1;
    if(nel[i] == 0)
        nel[i] = 1;
}
for(i = 0; i <= ntot; i++){
    if(nid[i] == 0)
        nid[i] = 1;
}
for(i = 0; i < nind; i++){ /* loop over individuals in pedigree */
    fprintf(fp2, "%d\t%d\n", nst[i], nel[i]+nst[i]-1);
}

for(i = 0; i < ntot; i++)
    fprintf(fp1, "%f\t%d\n", na[i], nid[i]);

free(ra);
free(na);
free(nid);
free(nel);
free(nst);
free(ja);
free(ia);

}
```

```
void minvert(double *D,int nind){

    int info;

    dpotrf('L', nind, D, nind, &info); /* computing inverse matrix */
    if(info!=0)
        printf("Error when calling function dpotrf\n");

    dpotri('L', nind, D, nind, &info);
    if(info!=0)
        printf("Error when calling function dpotri\n");

}

void mconv(float *D,double *D1,int nind){

    int i,j;

    for(i = 0; i < nind; i++)
        for(j = 0; j < nind; j++)
            D1[j*nind+i] = (double)D[j*nind+i];

}
```