#This code is largely identical to that published in Kang et al 2008, "Efficient control of population structure in model organism association mapping" (Genetics 178: 1709-1723) and available in the emma R package at (http://www.cs.ucla.edu/~eeskin/software.html). Modifications are indicated in commented out code.

emmat.test <- function(ys, xs, K, Z=NULL, X0 = NULL, ngrids=100, llim=-10, ulim=10, esp=1e-10, ponly = FALSE) {

if ( is.null(dim(ys)) || ncol(ys) == 1 ) {

ys <- matrix(ys,1,length(ys))

}

if ( is.null(dim(xs)) || ncol(xs) == 1 ) {

xs <- matrix(xs,1,length(xs))

}

if ( is.null(X0) ) {

X0 <- matrix(1,ncol(ys),1)

}

g <- nrow(ys)

n <- ncol(ys)

m <- nrow(xs)

t <- ncol(xs)

q0 <- ncol(X0)

q1 <- q0 + m

stopifnot(nrow(K) == t)

stopifnot(ncol(K) == t)

stopifnot(nrow(X0) == n)

if ( !ponly ) {

REMLs <- matrix(nrow=m,ncol=g)

vgs <- matrix(nrow=m,ncol=g)

ves <- matrix(nrow=m,ncol=g)

}

dfs <- matrix(nrow=m,ncol=g)

stats <- matrix(nrow=m,ncol=g)

#JT: added a second and third stats matrix (for recovering stats if running a change point model)

stats2 <- matrix(nrow=m,ncol=g)

stats3 <- matrix(nrow=m,ncol=g)

ps <- matrix(nrow=m,ncol=g)

#JT: added a second and third p matrix (for recovering p values for additional rank betas if running a change point model)

ps2 <- matrix(nrow=m,ncol=g)

ps3 <- matrix(nrow=m,ncol=g)

#JT: added betas here

betas<-matrix(nrow=g,ncol=q1)

if ( sum(is.na(ys)) == 0 ) {

eig.L <- emma.eigen.L(Z,K)

x.prev <- vector(length=0)

#JT: changed from i in 1:m so I could run both components of a change point model in 1 run if necessary

for(i in 1:1) {

vids <- !is.na(xs[i,])

nv <- sum(vids)

xv <- xs[i,vids]

# if ( ( mean(xv) <= 0 ) || ( mean(xv) >= 1 ) ) {

# if ( !ponly ) {

# vgs[i,] <- rep(NA,g)

# ves[i,] <- rep(NA,g)

# dfs[i,] <- rep(NA,g)

# REMLs[i,] <- rep(NA,g)

# stats[i,] <- rep(NA,g)

# }

# ps[i,] = rep(1,g)

#

# }

# else if ( identical(x.prev, xv) ) {

# if ( !ponly ) {

# vgs[i,] <- vgs[i-1,]

# ves[i,] <- ves[i-1,]

# dfs[i,] <- dfs[i-1,]

# REMLs[i,] <- REMLs[i-1,]

# stats[i,] <- stats[i-1,]

# }

# ps[i,] <- ps[i-1,]

# }

# else {

if ( is.null(Z) ) {

#JT: altered the code here so that X includes both rank columns around the change point--did not specify no missing data w/vids because I'm assuming there are no missing data in these files

X <- cbind(X0[vids,,drop=FALSE],t(xs))

eig.R1 = emma.eigen.R.wo.Z(K[vids,vids],X)

}

else {

vrows <- as.logical(rowSums(Z[,vids]))

X <- cbind(X0[vrows,,drop=FALSE],Z[vrows,vids,drop=FALSE]%\*%t(xs[i,vids,drop=FALSE]))

eig.R1 = emma.eigen.R.w.Z(Z[vrows,vids],K[vids,vids],X)

}

for(j in 1:g) {

if ( nv == t ) {

REMLE <- emma.REMLE(ys[j,],X,K,Z,ngrids,llim,ulim,esp,eig.R1)

if ( is.null(Z) ) {

U <- eig.L$vectors \* matrix(sqrt(1/(eig.L$values+REMLE$delta)),t,t,byrow=TRUE)

dfs[i,j] <- nv - q1

}

else {

U <- eig.L$vectors \* matrix(c(sqrt(1/(eig.L$values+REMLE$delta)),rep(sqrt(1/REMLE$delta),n-t)),n,n,byrow=TRUE)

dfs[i,j] <- n - q1

}

yt <- crossprod(U,ys[j,])

Xt <- crossprod(U,X)

#JT: modified here again to take pseudoinverses (for increased numerical stability)

iXX <- pseudoinverse(crossprod(Xt,Xt),1e-9)

# iXX <- solve(crossprod(Xt,Xt))

beta <- iXX%\*%crossprod(Xt,yt)

if ( !ponly ) {

vgs[i,j] <- REMLE$vg

ves[i,j] <- REMLE$ve

REMLs[i,j] <- REMLE$REML

}

#JT: added stats2 and stats3 here in order to get statistics and p values on the second beta for rank if running change point models

stats[i,j] <- beta[q1]/sqrt(iXX[q1,q1]\*REMLE$vg)

#CHECK THIS, IS THIS STATISTIC CORRECT???

stats2[i,j] <- beta[q1-1]/sqrt(iXX[q1-1,q1-1]\*REMLE$vg)

if(ncol(xo) >= 3){

stats3[i,j] <- beta[q1-2]/sqrt(iXX[q1-2,q1-2]\*REMLE$vg)

}

#JT: added here

betas[j,]<-beta

}

else {

if ( is.null(Z) ) {

eig.L0 <- emma.eigen.L.wo.Z(K[vids,vids])

nr <- sum(vids)

yv <- ys[j,vids]

REMLE <- emma.REMLE(yv,X,K[vids,vids,drop=FALSE],NULL,ngrids,llim,ulim,esp,eig.R1)

U <- eig.L0$vectors \* matrix(sqrt(1/(eig.L0$values+REMLE$delta)),nr,nr,byrow=TRUE)

dfs[i,j] <- nr - q1

}

else {

eig.L0 <- emma.eigen.L.w.Z(Z[vrows,vids,drop=FALSE],K[vids,vids])

yv <- ys[j,vrows]

nr <- sum(vrows)

tv <- sum(vids)

REMLE <- emma.REMLE(yv,X,K[vids,vids,drop=FALSE],Z[vrows,vids,drop=FALSE],ngrids,llim,ulim,esp,eig.R1)

U <- eig.L0$vectors \* matrix(c(sqrt(1/(eig.L0$values+REMLE$delta)),rep(sqrt(1/REMLE$delta),nr-tv)),nr,nr,byrow=TRUE)

dfs[i,j] <- nr - q1

}

yt <- crossprod(U,yv)

Xt <- crossprod(U,X)

iXX <- solve(crossprod(Xt,Xt))

beta <- iXX%\*%crossprod(Xt,yt)

if (!ponly) {

vgs[i,j] <- REMLE$vg

ves[i,j] <- REMLE$ve

REMLs[i,j] <- REMLE$REML

}

stats[i,j] <- beta[q1]/sqrt(iXX[q1,q1]\*REMLE$vg)

#JT: added here

betas[j,]<-beta

}

# }

ps[i,] <- 2\*pt(abs(stats[i,]),dfs[i,],lower.tail=FALSE)

#JT: added second and third set of ps here for additional linear models

ps2[i,] <- 2\*pt(abs(stats2[i,]),dfs[i,],lower.tail=FALSE)

if(ncol(xo) >= 3){

ps3[i,] <- 2\*pt(abs(stats3[i,]),dfs[i,],lower.tail=FALSE)

}

}

}

}

else {

eig.L <- emma.eigen.L(Z,K)

eig.R0 <- emma.eigen.R(Z,K,X0)

x.prev <- vector(length=0)

for(i in 1:m) {

vids <- !is.na(xs[i,])

nv <- sum(vids)

xv <- xs[i,vids]

if ( ( mean(xv) <= 0 ) || ( mean(xv) >= 1 ) ) {

if (!ponly) {

vgs[i,] <- rep(NA,g)

ves[i,] <- rep(NA,g)

REMLs[i,] <- rep(NA,g)

dfs[i,] <- rep(NA,g)

}

ps[i,] = rep(1,g)

}

else if ( identical(x.prev, xv) ) {

if ( !ponly ) {

stats[i,] <- stats[i-1,]

vgs[i,] <- vgs[i-1,]

ves[i,] <- ves[i-1,]

REMLs[i,] <- REMLs[i-1,]

dfs[i,] <- dfs[i-1,]

}

ps[i,] = ps[i-1,]

}

else {

if ( is.null(Z) ) {

X <- cbind(X0,xs[i,])

if ( nv == t ) {

eig.R1 = emma.eigen.R.wo.Z(K,X)

}

}

else {

vrows <- as.logical(rowSums(Z[,vids,drop=FALSE]))

X <- cbind(X0,Z[,vids,drop=FALSE]%\*%t(xs[i,vids,drop=FALSE]))

if ( nv == t ) {

eig.R1 = emma.eigen.R.w.Z(Z,K,X)

}

}

for(j in 1:g) {

vrows <- !is.na(ys[j,])

if ( nv == t ) {

yv <- ys[j,vrows]

nr <- sum(vrows)

if ( is.null(Z) ) {

if ( nr == n ) {

REMLE <- emma.REMLE(yv,X,K,NULL,ngrids,llim,ulim,esp,eig.R1)

U <- eig.L$vectors \* matrix(sqrt(1/(eig.L$values+REMLE$delta)),n,n,byrow=TRUE)

}

else {

eig.L0 <- emma.eigen.L.wo.Z(K[vrows,vrows,drop=FALSE])

REMLE <- emma.REMLE(yv,X[vrows,,drop=FALSE],K[vrows,vrows,drop=FALSE],NULL,ngrids,llim,ulim,esp)

U <- eig.L0$vectors \* matrix(sqrt(1/(eig.L0$values+REMLE$delta)),nr,nr,byrow=TRUE)

}

dfs[i,j] <- nr-q1

}

else {

if ( nr == n ) {

REMLE <- emma.REMLE(yv,X,K,Z,ngrids,llim,ulim,esp,eig.R1)

U <- eig.L$vectors \* matrix(c(sqrt(1/(eig.L$values+REMLE$delta)),rep(sqrt(1/REMLE$delta),n-t)),n,n,byrow=TRUE)

}

else {

vtids <- as.logical(colSums(Z[vrows,,drop=FALSE]))

eig.L0 <- emma.eigen.L.w.Z(Z[vrows,vtids,drop=FALSE],K[vtids,vtids,drop=FALSE])

REMLE <- emma.REMLE(yv,X[vrows,,drop=FALSE],K[vtids,vtids,drop=FALSE],Z[vrows,vtids,drop=FALSE],ngrids,llim,ulim,esp)

U <- eig.L0$vectors \* matrix(c(sqrt(1/(eig.L0$values+REMLE$delta)),rep(sqrt(1/REMLE$delta),nr-sum(vtids))),nr,nr,byrow=TRUE)

}

dfs[i,j] <- nr-q1

}

yt <- crossprod(U,yv)

Xt <- crossprod(U,X[vrows,,drop=FALSE])

iXX <- solve(crossprod(Xt,Xt))

beta <- iXX%\*%crossprod(Xt,yt)

if ( !ponly ) {

vgs[i,j] <- REMLE$vg

ves[i,j] <- REMLE$ve

REMLs[i,j] <- REMLE$REML

}

stats[i,j] <- beta[q1]/sqrt(iXX[q1,q1]\*REMLE$vg)

}

else {

if ( is.null(Z) ) {

vtids <- vrows & vids

eig.L0 <- emma.eigen.L.wo.Z(K[vtids,vtids,drop=FALSE])

yv <- ys[j,vtids]

nr <- sum(vtids)

REMLE <- emma.REMLE(yv,X[vtids,,drop=FALSE],K[vtids,vtids,drop=FALSE],NULL,ngrids,llim,ulim,esp)

U <- eig.L0$vectors \* matrix(sqrt(1/(eig.L0$values+REMLE$delta)),nr,nr,byrow=TRUE)

Xt <- crossprod(U,X[vtids,,drop=FALSE])

dfs[i,j] <- nr-q1

}

else {

vtids <- as.logical(colSums(Z[vrows,,drop=FALSE])) & vids

vtrows <- vrows & as.logical(rowSums(Z[,vids,drop=FALSE]))

eig.L0 <- emma.eigen.L.w.Z(Z[vtrows,vtids,drop=FALSE],K[vtids,vtids,drop=FALSE])

yv <- ys[j,vtrows]

nr <- sum(vtrows)

REMLE <- emma.REMLE(yv,X[vtrows,,drop=FALSE],K[vtids,vtids,drop=FALSE],Z[vtrows,vtids,drop=FALSE],ngrids,llim,ulim,esp)

U <- eig.L0$vectors \* matrix(c(sqrt(1/(eig.L0$values+REMLE$delta)),rep(sqrt(1/REMLE$delta),nr-sum(vtids))),nr,nr,byrow=TRUE)

Xt <- crossprod(U,X[vtrows,,drop=FALSE])

dfs[i,j] <- nr-q1

}

yt <- crossprod(U,yv)

iXX <- solve(crossprod(Xt,Xt))

beta <- iXX%\*%crossprod(Xt,yt)

if ( !ponly ) {

vgs[i,j] <- REMLE$vg

ves[i,j] <- REMLE$ve

REMLs[i,j] <- REMLE$REML

}

stats[i,j] <- beta[q1]/sqrt(iXX[q1,q1]\*REMLE$vg)

}

}

ps[i,] <- 2\*pt(abs(stats[i,]),dfs[i,],lower.tail=FALSE)

}

}

}

if ( ponly ) {

return (ps)

}

else {

#modified return to get the other set of ps and stats if running a change point model

return (list(ps=ps,ps2=ps2,ps3=ps3,REMLs=REMLs,stats=stats,stats2=stats2,stats3=stats3,dfs=dfs,vgs=vgs,ves=ves,betas=betas))

}

}

emma.eigen.L <- function(Z,K,complete=TRUE) {

if ( is.null(Z) ) {

return(emma.eigen.L.wo.Z(K))

}

else {

return(emma.eigen.L.w.Z(Z,K,complete))

}

}

emma.eigen.L.wo.Z <- function(K) {

eig <- eigen(K,symmetric=TRUE)

#added here

n<-as.numeric(eig$values>1e-4)

eig$values<-eig$values\*n

return(list(values=eig$values,vectors=eig$vectors))

}

emma.eigen.R.wo.Z <- function(K, X) {

n <- nrow(X)

q <- ncol(X)

#JT: modified the line here, requires the package "corpcor" to perform pseudoinverses instead of using solve

# S <- diag(n)-X%\*%solve(crossprod(X,X))%\*%t(X)

S <- diag(n)-X%\*%pseudoinverse(crossprod(X,X),1e-4)%\*%t(X)

eig <- eigen(S%\*%(K+diag(1,n))%\*%S,symmetric=TRUE)

stopifnot(!is.complex(eig$values))

#added

bad<-as.numeric(eig$values>1e-4)

eig$values<-eig$values\*bad

return(list(values=eig$values[1:(n-q)]-1,vectors=eig$vectors[,1:(n-q)]))

}

emma.REMLE <- function(y, X, K, Z=NULL, ngrids=100, llim=-10, ulim=10,

esp=1e-10, eig.L = NULL, eig.R = NULL) {

n <- length(y)

t <- nrow(K)

q <- ncol(X)

# stopifnot(nrow(K) == t)

stopifnot(ncol(K) == t)

stopifnot(nrow(X) == n)

# if ( det(crossprod(X,X)) == 0 ) {

# warning("X is singular")

# return (list(REML=0,delta=0,ve=0,vg=0))

# }

if ( is.null(Z) ) {

if ( is.null(eig.R) ) {

eig.R <- emma.eigen.R.wo.Z(K,X)

}

etas <- crossprod(eig.R$vectors,y)

logdelta <- (0:ngrids)/ngrids\*(ulim-llim)+llim

m <- length(logdelta)

delta <- exp(logdelta)

Lambdas <- matrix(eig.R$values,n-q,m) + matrix(delta,n-q,m,byrow=TRUE)

Etasq <- matrix(etas\*etas,n-q,m)

LL <- 0.5\*((n-q)\*(log((n-q)/(2\*pi))-1-log(colSums(Etasq/Lambdas)))-colSums(log(Lambdas)))

dLL <- 0.5\*delta\*((n-q)\*colSums(Etasq/(Lambdas\*Lambdas))/colSums(Etasq/Lambdas)-colSums(1/Lambdas))

optlogdelta <- vector(length=0)

optLL <- vector(length=0)

if ( dLL[1] < esp ) {

optlogdelta <- append(optlogdelta, llim)

optLL <- append(optLL, emma.delta.REML.LL.wo.Z(llim,eig.R$values,etas))

}

if ( dLL[m-1] > 0-esp ) {

optlogdelta <- append(optlogdelta, ulim)

optLL <- append(optLL, emma.delta.REML.LL.wo.Z(ulim,eig.R$values,etas))

}

for( i in 1:(m-1) )

{

if ( ( dLL[i]\*dLL[i+1] < 0-esp\*esp ) && ( dLL[i] > 0 ) && ( dLL[i+1] < 0 ) )

{

r <- uniroot(emma.delta.REML.dLL.wo.Z, lower=logdelta[i], upper=logdelta[i+1], lambda=eig.R$values, etas=etas)

optlogdelta <- append(optlogdelta, r$root)

optLL <- append(optLL, emma.delta.REML.LL.wo.Z(r$root,eig.R$values, etas))

}

}

# optdelta <- exp(optlogdelta)

}

else {

if ( is.null(eig.R) ) {

eig.R <- emma.eigen.R.w.Z(Z,K,X)

}

etas <- crossprod(eig.R$vectors,y)

etas.1 <- etas[1:(t-q)]

etas.2 <- etas[(t-q+1):(n-q)]

etas.2.sq <- sum(etas.2\*etas.2)

logdelta <- (0:ngrids)/ngrids\*(ulim-llim)+llim

m <- length(logdelta)

delta <- exp(logdelta)

Lambdas <- matrix(eig.R$values,t-q,m) + matrix(delta,t-q,m,byrow=TRUE)

Etasq <- matrix(etas.1\*etas.1,t-q,m)

dLL <- 0.5\*delta\*((n-q)\*(colSums(Etasq/(Lambdas\*Lambdas))+etas.2.sq/(delta\*delta))/(colSums(Etasq/Lambdas)+etas.2.sq/delta)-(colSums(1/Lambdas)+(n-t)/delta))

optlogdelta <- vector(length=0)

optLL <- vector(length=0)

if ( dLL[1] < esp ) {

optlogdelta <- append(optlogdelta, llim)

optLL <- append(optLL, emma.delta.REML.LL.w.Z(llim,eig.R$values,etas.1,n,t,etas.2.sq))

}

if ( dLL[m-1] > 0-esp ) {

optlogdelta <- append(optlogdelta, ulim)

optLL <- append(optLL, emma.delta.REML.LL.w.Z(ulim,eig.R$values,etas.1,n,t,etas.2.sq))

}

for( i in 1:(m-1) )

{

if ( ( dLL[i]\*dLL[i+1] < 0-esp\*esp ) && ( dLL[i] > 0 ) && ( dLL[i+1] < 0 ) )

{

r <- uniroot(emma.delta.REML.dLL.w.Z, lower=logdelta[i], upper=logdelta[i+1], lambda=eig.R$values, etas.1=etas.1, n=n, t1=t, etas.2.sq = etas.2.sq )

optlogdelta <- append(optlogdelta, r$root)

optLL <- append(optLL, emma.delta.REML.LL.w.Z(r$root,eig.R$values, etas.1, n, t, etas.2.sq ))

}

}

# optdelta <- exp(optlogdelta)

}

maxdelta <- exp(optlogdelta[which.max(optLL)])

maxLL <- max(optLL)

if ( is.null(Z) ) {

maxva <- sum(etas\*etas/(eig.R$values+maxdelta))/(n-q)

}

else {

maxva <- (sum(etas.1\*etas.1/(eig.R$values+maxdelta))+etas.2.sq/maxdelta)/(n-q)

}

maxve <- maxva\*maxdelta

return (list(REML=maxLL,delta=maxdelta,ve=maxve,vg=maxva))

}

emma.delta.REML.LL.wo.Z <- function(logdelta, lambda, etas) {

nq <- length(etas)

delta <- exp(logdelta)

return( 0.5\*(nq\*(log(nq/(2\*pi))-1-log(sum(etas\*etas/(lambda+delta))))-sum(log(lambda+delta))) )

}

emma.delta.REML.dLL.wo.Z <- function(logdelta, lambda, etas) {

nq <- length(etas)

delta <- exp(logdelta)

etasq <- etas\*etas

ldelta <- lambda+delta

return( 0.5\*(nq\*sum(etasq/(ldelta\*ldelta))/sum(etasq/ldelta)-sum(1/ldelta)) )

}