Package 'NHMMfdr'

July 2, 2014

Type Package

Title Compute FDR under dependence using NHMM

Version 1.0.6

Date 2013-03-04

Author Pei Fen Kuan <peifen.kuan@stonybrook.edu>

Maintainer Pei Fen Kuan <peifen.kuan@stonybrook.edu>

Depends R (>= 2.10.0), MASS, locfdr

Description The NHMMfdr package implements the non-homogeneous Hidden Markov Model based FDR control as described in Kuan et al., 2011 for multiple comparison adjustment under dependence. It allows for prior information to be incorporated in the model to improve detection of significant tests.

License GPL (>= 2)

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-03-28 07:18:34

R topics documented:

`NHMMfdr-package` *Compute FDR under dependence using NHMM*

Description

The NHMMfdr package implements the non-homogeneous Hidden Markov Model based FDR control as described in Kuan et al., 2011 for multiple comparison adjustment under dependence. It allows for prior information to be incorporated in the model to improve detection of significant tests.

Details

Package:	NHMMfdr
Type:	Package
Version:	1.0.5
Date:	2013-03-04
License:	GPL (>= 2)
LazyLoad:	yes

Author(s)

Pei Fen Kuan <peifen.kuan@stonybrook.edu>

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

See Also

`fdr.nhmm`, `LIS.adjust`

Examples

```
library(NHMMfdr)

################
# Simulate data
################

### simulate covariate and transition prob
NUM1 <- 1000
Z <- rnorm(NUM1)
Z <- matrix(Z,ncol=1)

Z <- apply(Z,2,scale)
trans.par1.true <- c(0,0,0,0)

trans.par2.true <- rnorm(3+dim(Z)[2])

print(trans.par2.true[-1])

A.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
dist.included=FALSE)$A
pii.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
  dist.included=FALSE)$pii

### the null distribution
f0 <- c(0, 1)

### the alternative distribution
f1 <- c(3, 1)

### the NHMM data

simdat <- simdata.nhmm(NUM1, pii.true, A.true, f0, 1, f1)

### the observed values
x <- simdat$o

### the unobserved true states
theta1 <- simdat$s

################
# Model fitting
################

fit.nhmm <- fdr.nhmm(x, Z, dist = NULL, log.transform.dist = FALSE,
alttype ='mixnormal', L=1, maxiter = 100, nulltype = 0, modeltype = 'NHMM',
epsilon=1e-4)

### checking estimated parameters
print(fit.nhmm$trans.par2[-1])
```

```
################
# Adjust LIS
################

LIS.adjust <- LIS.adjust(fit.nhmm$LIS, fdr = 0.1, adjust = TRUE)

### tests which are statistically significant

sig.test <- which(LIS.adjust$States == 1)
length(sig.test)
```

`compute.A.nhmm` *Compute transition probability (internal function)*

Description

An internal function to be used by `fdr.nhmm`.

Usage

```
compute.A.nhmm(Z, trans.par1, trans.par2, dist.included = TRUE)
```

Arguments

`Z`	A matrix of covariates
`trans.par1`	Transition parameters for State 0
`trans.par2`	Transition parameters for State 1
`dist.included`	Logical value. `dist.included = TRUE` if probe spacing/distance is included in the first column of `Z`.

Value

Returns transition probability matrix and initial probabilities.

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

See Also

`fdr.nhmm`, `LIS.adjust`

Examples

```
library(NHMMfdr)

################
# Simulate data
################

### simulate covariate and transition prob
NUM1 <- 1000
Z <- rnorm(NUM1)
Z <- matrix(Z,ncol=1)

Z <- apply(Z,2,scale)
trans.par1.true <- c(0,0,0,0)

trans.par2.true <- rnorm(3+dim(Z)[2])

print(trans.par2.true[-1])

A.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
dist.included=FALSE)$A
pii.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
  dist.included=FALSE)$pii

### the null distribution
f0 <- c(0, 1)

### the alternative distribution
f1 <- c(3, 1)

### the NHMM data

simdat <- simdata.nhmm(NUM1, pii.true, A.true, f0, 1, f1)

### the observed values
x <- simdat$o

### the unobserved true states
theta1 <- simdat$s

################
# Model fitting
################

fit.nhmm <- fdr.nhmm(x, Z, dist = NULL, log.transform.dist = FALSE,
alttype ='mixnormal', L=1, maxiter = 100, nulltype = 0, modeltype = 'NHMM', epsilon=1e-4)

### checking estimated parameters
print(fit.nhmm$trans.par2[-1])

################
```

```
# Adjust LIS
################

LIS.adjust <- LIS.adjust(fit.nhmm$LIS, fdr = 0.1, adjust = TRUE)

### tests which are statistically significant

sig.test <- which(LIS.adjust$States == 1)
length(sig.test)
```

---

compute.gradient *Intermediate function*

Description

Intermediate function

Usage

```
compute.gradient(Z, dgamma, gamma, trans.par1, trans.par2, dist.included = TRUE)
```

Arguments

Z

dgamma

gamma

trans.par1

trans.par2

dist.included

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(Z, dgamma, gamma, trans.par1, trans.par2, dist.included=TRUE){

gradient <- rep(0,3+dim(Z)[2])

tmp.trans.prob <- compute.A.nhmm(Z, trans.par1, trans.par2, dist.included=dist.included)

pii <- tmp.trans.prob$pii
A <- tmp.trans.prob$A

gradient[1] <- gamma[1,2] - pii[2]
gradient[2] <- sum(dgamma[1,2,] - gamma[-dim(gamma)[1],1]*A[1,2,])
gradient[3] <- sum(dgamma[2,2,] - gamma[-dim(gamma)[1],2]*A[2,2,])

tmp <- rep(0,dim(dgamma)[3])

for(i in 1:2){
tmp <- tmp + gamma[-dim(Z)[1],i]*A[i,2,]
}
gradient[-c(1:3)] <- (gamma[1,2] - pii[2])*Z[1,]
+ apply(matrix((gamma[-1,2] - tmp)*Z[-1,],ncol=dim(Z)[2]),2,sum)

        if(dist.included==TRUE){
          gradient[4] <- (gamma[1,2] - pii[2])*Z[1,1] + sum(dgamma[1,2,]*Z[-1,1])
          - sum(dgamma[2,2,]*Z[-1,1]) - sum(gamma[-dim(Z)[1],1]*A[1,2,]*Z[-1,1])
          + sum(gamma[-dim(Z)[1],2]*A[2,2,]*Z[-1,1])
        }

return(-gradient)

}
```

---

em.hmm *Intermediate function*

Description

Intermediate function

Usage

```
em.hmm(x, alttype = "mixnormal", L = 2, maxiter = 1000, nulltype = 2, symmetric = FALSE,
epsilon = 1e-04)
```

Arguments

x

alttype

L

maxiter

nulltype

symmetric

epsilon

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, alttype='mixnormal', L=2, maxiter=1000, nulltype=2, symmetric = FALSE,
epsilon=1e-4)
{

NUM<-length(x)

ptol<-epsilon

niter<-0

# Assuming it will converge
converged=TRUE

### initializing model parameters

if(alttype == 'kernel'){

### initializing model parameters
```

```
pii.new <- c(0.5, 0.5)
A.new <- array(c(0.8, 0.4, 0.2, 0.6),c(2,2,NUM - 1))

f0.new<-c(0, 1)
locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

f1.new <- 0.5*dnorm(x,4,1)+0.5*dnorm(x,-4,1)

diff<-1

### The E-M Algorithm

while(diff>ptol && niter<maxiter)
{

niter<-niter+1

pii.old <- pii.new
A.old <- A.new
f0.old <- f0.new
f1.old <- f1.new

## updating the weights and probabilities of hidden states

forwardbackward.res <- forwardbackward1.kernel(x, pii.old, A.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
dgamma <- forwardbackward.res$ts
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

for (i in 0:1)
{
  pii.new[i+1] <- gamma[1, i+1]
}


for (i in 0:1)
{
  for (j in 0:1)
  {
    q1 <- sum(dgamma[i+1, j+1, ])
```

```
    q2 <- sum(gamma[1:(NUM-1), i+1])
    A.new[i+1, j+1,] <- q1/q2
  }
}

q5 <- sum(gamma[, 1]*x)
mu0 <- q5/sum(gamma[, 1])

q6 <- sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0 <- sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

if(symmetric == FALSE){
kern.f1 <- density(x,weights=gamma[,2]/sum(gamma[,2]))
f1.new <- approx(kern.f1$x, kern.f1$y, x, rule = 2, ties="ordered")$y
}

if(symmetric == TRUE){
kern.f1 <- density(c(x,2*f0.new[1]-x),
weights=c(gamma[,2],gamma[,2])/sum(c(gamma[,2],gamma[,2])))
f1.new <- approx(kern.f1$x, kern.f1$y, x, rule = 2, ties="ordered")$y
}

df1<-abs(A.old-A.new)
df2<-abs(f1.old-f1.new)
diff<-max(df1, df2)

if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

 lfdr<-gamma[, 1]
if (converged) {
 logL<--sum(log(c0))
if (nulltype > 0) {
BIC<-logL-(3+4-2)*log(NUM)/2
} else {
BIC<-logL-(3+2-2)*log(NUM)/2
}
```

```
 em.var <- list(pii=pii.new, A=A.new[,,1], f0=f0.new, f1= kern.f1, LIS=lfdr, logL=logL,
 BIC=BIC, ni=niter, converged=converged)
} else {
 BIC <- logL <- (-Inf)
 em.var <- list(pii=pii.old, A=A.old[,,1], f0=f0.old, f1= kern.f1, LIS=lfdr, logL=logL,
 BIC=BIC, ni=niter, converged=converged)
}
}

if(alttype == 'mixnormal'){
########
# L=1
########

if (L==1)
{

pii.new <- c(0.5, 0.5)
A.new <- array(c(0.8, 0.4, 0.6, 0.2),c(2,2,NUM - 1))

f0.new<-c(0, 1)
locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

f1.new <- c(4, 1)

diff<-1

### The E-M Algorithm

while(diff>ptol && niter<maxiter)
{

niter<-niter+1

pii.old <- pii.new
A.old <- A.new
f0.old <- f0.new
f1.old <- f1.new

## updating the weights and probabilities of hidden states

forwardbackward.res <- forwardbackward1(x, pii.old, A.old, f0.old, f1.old)
```

```
gamma <- forwardbackward.res$pr
dgamma <- forwardbackward.res$ts
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

for (i in 0:1)
{
  pii.new[i+1] <- gamma[1, i+1]
}


for (i in 0:1)
{
  for (j in 0:1)
  {
    q1 <- sum(dgamma[i+1, j+1, ])
    q2 <- sum(gamma[1:(NUM-1), i+1])
    A.new[i+1, j+1,] <- q1/q2
  }
}

q5 <- sum(gamma[, 1]*x)
mu0 <- q5/sum(gamma[, 1])

q6 <- sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0 <- sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

q1 <- sum(gamma[, 2])
q2 <- sum(gamma[, 2]*x)
mu1 <- q2/q1
q3 <- sum(gamma[, 2]*(x-mu1)*(x-mu1))
sd1 <- sqrt(q3/q1)
f1.new <- c(mu1, sd1)

df1<-abs(A.old-A.new)
df2<-abs(f1.old-f1.new)
diff<-max(df1, df2)

if (is.na(diff)) {
   converged=FALSE;
```

```
   break;
}

}

 lfdr<-gamma[, 1]
if (converged) {
 logL<--sum(log(c0))
if (nulltype > 0) {
BIC<-logL-(3*L+4)*log(NUM)/2
} else {
BIC<-logL-(3*L+2)*log(NUM)/2
}

 em.var <- list(pii=pii.new, A=A.new[,,1], f0=f0.new, f1=f1.new, LIS=lfdr, logL=logL,
 BIC=BIC, ni=niter, converged=converged)
} else {
 BIC <- logL <- (-Inf)
 em.var <- list(pii=pii.old, A=A.old[,,1], f0=f0.old, f1=f1.old, LIS=lfdr, logL=logL,
 BIC=BIC, ni=niter, converged=converged)
}

}

#######
# L>1
#######

else if (L>1)
{

pii.new<-c(0.5, 0.5)
A.new <- array(c(0.95, 0.5, 0.05, 0.5),c(2,2,NUM - 1))

f0.new<-c(0, 1)
locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

pc.new <- rep(1, L)/L
mus <- seq(from=-1, by=1.5, length=L)
sds <- rep(1, L)
f1.new <- cbind(mus, sds)

diff <- 1
```

```
### The E-M Algorithm

while(diff>ptol && niter<maxiter)
{

niter <- niter+1

pii.old <- pii.new
A.old <- A.new
pc.old <- pc.new
f0.old <- f0.new
f1.old <- f1.new

## updating the weights and probabilities of hidden states

forwardbackward.res <- forwardbackward(x, pii.old, A.old, pc.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
dgamma <- forwardbackward.res$ts
omega <- forwardbackward.res$wt
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

for (i in 0:1)
{
  pii.new[i+1] <- gamma[1, i+1]
}


for (i in 0:1)
{
  for (j in 0:1)
  {
    q1 <- sum(dgamma[i+1, j+1, ])
    q2 <- sum(gamma[1:(NUM-1), i+1])
    A.new[i+1, j+1,] <- q1/q2
  }
}

q5 <- sum(gamma[, 1]*x)
mu0 <- q5/sum(gamma[, 1])

q6 <- sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0 <- sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
```

```
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

mus <- 1:L
sds <- 1:L

for (c in 1:L)
{
  q1 <- sum(omega[, c])
  q2 <- sum(gamma[, 2])
  pc.new[c] <- q1/q2


  q3 <- sum(omega[, c]*x)
  mus[c] <- q3/q1

  q4 <- sum(omega[, c]*(x-mus[c])*(x-mus[c]))
  sds[c] <- sqrt(q4/q1)

}

f1.new <- cbind(mus, sds)

df1 <- abs(A.old-A.new)
df2 <- abs(f1.old-f1.new)
diff <- max(df1, df2)

if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

lfdr<-gamma[, 1]
if (converged) {
 logL <- -sum(log(c0))
if (nulltype > 0) {
BIC<-logL-(3*L+4)*log(NUM)/2
} else {
BIC<-logL-(3*L+2)*log(NUM)/2
}
 em.var <- list(pii=pii.new, A=A.new[,,1], pc=pc.new, f0=f0.new, f1=f1.new, LIS=lfdr,
 logL=logL, BIC=BIC, ni=niter, converged=converged)
} else {
 logL <- (-Inf)
 BIC <- logL <- (-Inf)
 em.var <- list(pii=pii.old, A=A.old[,,1], pc=pc.old, f0=f0.old, f1=f1.old, LIS=lfdr,
 logL=logL, BIC=BIC, ni=niter, converged=converged)
}
```

```
}
}

return (em.var)
}
```

---

em.indep *Intermediate function*

Description

Intermediate function

Usage

```
em.indep(x, alttype = "mixnormal", L = 2, maxiter = 1000, nulltype = 2,
symmetric = FALSE,epsilon = 1e-04)
```

Arguments

x

alttype

L

maxiter

nulltype

symmetric

epsilon

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, alttype='mixnormal', L=2, maxiter=1000, nulltype = 2, symmetric = FALSE,
epsilon=1e-4)
{

NUM<-length(x)

ptol<-epsilon
niter<-0

# Assuming it will converge
converged=TRUE

### initializing model parameters
if(alttype == 'kernel'){

f0.new<-c(2, 1)

locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

f1.new <- 0.5*dnorm(x,4,1)+0.5*dnorm(x,-4,1)
ptheta.new <- c(0.5,0.5)

diff<-1

### The E-M Algorithm

while(diff>ptol && niter<maxiter)
{

niter<-niter+1

ptheta.old <- ptheta.new
f0.old <- f0.new
f1.old <- f1.new

## updating the weights and probabilities of hidden states
```

```
forwardbackward.res <- forwardbackward1.indep.kernel(x, ptheta.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

ptheta.new <- apply(gamma,2,sum)/NUM

q5 <- sum(gamma[, 1]*x)
mu0 <- q5/sum(gamma[, 1])

q6 <- sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0 <- sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

if(symmetric == FALSE){
kern.f1 <- density(x,weights=gamma[,2]/sum(gamma[,2]))
f1.new <- approx(kern.f1$x, kern.f1$y, x, rule = 2, ties="ordered")$y
}

if(symmetric == TRUE){
kern.f1 <- density(c(x,2*f0.new[1]-x),
weights=c(gamma[,2],gamma[,2])/sum(c(gamma[,2],gamma[,2])))
f1.new <- approx(kern.f1$x, kern.f1$y, x, rule = 2, ties="ordered")$y
}

df2 <- abs(f1.old-f1.new)
diff <- max(df2)

if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

 lfdr <- gamma[, 1]
if (converged) {
 logL<- sum(log(c0))
if (nulltype > 0) {
BIC <- logL-(3+2-2)*log(NUM)/2
```

```
} else {
BIC <- logL-(3-2)*log(NUM)/2
}

 em.var<-list(ptheta=ptheta.new, f0=f0.new, f1=kern.f1, LIS=lfdr, logL=logL, BIC=BIC,
 ni=niter, converged=converged)
} else {
 BIC<- logL<- (-Inf)
 em.var<-list(ptheta=ptheta.old, f0=f0.old, f1=kern.f1, LIS=lfdr, logL=logL, BIC=BIC,
 ni=niter, converged=converged)
}

}

if(alttype == 'mixnormal'){

########
# L=1
########

if (L==1)
{

f0.new<-c(2, 1)

locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

f1.new<-c(4, 1)
ptheta.new <- c(0.5,0.5)

diff<-1

### The E-M Algorithm

while(diff>ptol && niter<maxiter)
{

niter<-niter+1

ptheta.old <- ptheta.new
f0.old <- f0.new
f1.old <- f1.new

## updating the weights and probabilities of hidden states
```

```
forwardbackward.res <- forwardbackward1.indep(x, ptheta.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

ptheta.new <- apply(gamma,2,sum)/NUM

q5 <- sum(gamma[, 1]*x)
mu0 <- q5/sum(gamma[, 1])

q6 <- sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0 <- sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

q1 <- sum(gamma[, 2])
q2 <- sum(gamma[, 2]*x)
mu1 <- q2/q1
q3 <- sum(gamma[, 2]*(x-mu1)*(x-mu1))
sd1 <- sqrt(q3/q1)
f1.new <- c(mu1, sd1)

df2 <- abs(f1.old-f1.new)
diff <- max(df2)

if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

 lfdr <- gamma[, 1]
if (converged) {
 logL<- sum(log(c0))
if (nulltype > 0) {
BIC<-logL-(3*L+2)*log(NUM)/2
} else {
BIC<-logL-(3*L)*log(NUM)/2
}
```

```
 em.var<-list(ptheta=ptheta.new, f0=f0.new, f1=f1.new, LIS=lfdr, logL=logL, BIC=BIC,
 ni=niter, converged=converged)
} else {
 BIC<- logL<- (-Inf)
 em.var<-list(ptheta=ptheta.old, f0=f0.old, f1=f1.old, LIS=lfdr, logL=logL, BIC=BIC,
 ni=niter, converged=converged)
}

}

#######
# L>1
#######

else if (L>1)
{

ptheta.new <- c(0.5, 0.5)
pc.new <- rep(1, L)/L
mus <- seq(from=-1, by=1.5, length=L)
sds <- rep(1, L)
f0.new <- c(2, 1)

locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}


f1.new<-cbind(mus, sds)

diff<-1

### The E-M Algorithm

while(diff>ptol && niter<maxiter)
{

niter <- niter+1

ptheta.old <- ptheta.new
pc.old <- pc.new
f0.old <- f0.new
f1.old <- f1.new

## updating the weights and probabilities of hidden states
```

```
forwardbackward.res <- forwardbackward.indep(x, ptheta.old, pc.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
omega <- forwardbackward.res$wt
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

ptheta.new <- apply(gamma,2,sum)/NUM

q5 <- sum(gamma[, 1]*x)
mu0 <- q5/sum(gamma[, 1])

q6 <- sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0 <- sqrt(q6/sum(gamma[, 1]))

f0.new <- c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}


mus <- 1:L
sds <- 1:L

for (c in 1:L)
{

  q1 <- sum(omega[, c])
  q2 <- sum(gamma[, 2])
  pc.new[c] <- q1/q2

  q3 <- sum(omega[, c]*x)
  mus[c] <- q3/q1

  q4 <- sum(omega[, c]*(x-mus[c])*(x-mus[c]))
  sds[c] <- sqrt(q4/q1)

}

f1.new <- cbind(mus, sds)

df2 <- abs(f1.old-f1.new)
diff <- max(df2)
```

```
if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

lfdr <- gamma[, 1]
if (converged) {
 logL <- sum(log(c0))
if (nulltype > 0) {
BIC <- logL-(3*L+2)*log(NUM)/2
} else {
BIC <- logL-(3*L)*log(NUM)/2
}
 em.var<-list(ptheta=ptheta.new, pc=pc.new, f0=f0.new, f1=f1.new, LIS=lfdr, logL=logL,
 BIC=BIC, ni=niter, converged=converged)
} else {
 logL <- (-Inf)
 BIC <- logL<- (-Inf)
 em.var <-list(ptheta=ptheta.old, pc=pc.old, f0=f0.old, f1=f1.old, LIS=lfdr, logL=logL,
 BIC=BIC, ni=niter, converged=converged)
}

}
}

return (em.var)
}
```

---

em.nhmm *Intermediate function*

Description

Intermediate function

Usage

```
em.nhmm(x, Z, dist, dist.included = TRUE, alttype = "mixnormal", L = 2, maxiter = 1000,
nulltype = 2,symmetric = FALSE, epsilon = 1e-04)
```

Arguments

x

Z

dist

dist.included

alttype

L

maxiter

nulltype

symmetric

epsilon

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, Z, dist, dist.included=TRUE, alttype='mixnormal', L=2, maxiter=1000,
nulltype=2, symmetric=FALSE, epsilon=1e-4)
{

NUM<-length(x)

ptol<-epsilon

niter<-0

if(length(dist)>0&length(Z)>0){
  z1 <- length(dist)
  if(is.vector(Z)==TRUE){
    z2 <- length(Z)
  }else z2 <- dim(Z)[1]

  if(z1!=z2){
    cat('Error: x and Z not compatible','\n')
    stop
  }
}
```

```
if(length(Z)>0){
if(is.vector(Z)==TRUE) Z <- matrix(Z,ncol=1)
if(dim(Z)[1]!=NUM){
cat('Error: x and Z not compatible','\n')
stop
}
if(dim(Z)[2] == 1) Z <- scale(Z)
if(dim(Z)[2] > 1) Z <- apply(Z,2,scale)
cat('Scaling covariates','\n')
}

Z <- cbind(dist,Z)
Z_all <- matrix(0,ncol=dim(Z)[2],nrow=(NUM-1)*2)
for(i in 1:dim(Z)[2]){
Z_all[,i] <- rep(Z[-1,i],each=2)
}

# Assuming it will converge
converged=TRUE

if(alttype == 'kernel'){

tmp.trans.update <- 0
diff<-1
logL.iter <- 0

while(diff>ptol && niter<maxiter)
{

### control for error in update_trans.prob.nhmm ###

if(length(tmp.trans.update) == 1){

diff<-1
logL.iter <- 0
niter <- 0

trans.par1.new <- c(0,0,0,0)
trans.par2.new <- rnorm(3+dim(Z)[2])
trans.par2.new[4] <- abs(trans.par2.new[4])

tmp.trans.prob <- compute.A.nhmm (Z, trans.par1.new, trans.par2.new,
dist.included=dist.included)
pii.new <- tmp.trans.prob$pii
A.new <- tmp.trans.prob$A

f0.new<-c(0, 1)
locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
```

```
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}
f1.new <- 0.5*dnorm(x,4,1)+0.5*dnorm(x,-4,1)
}

### The E-M Algorithm

niter<-niter+1

trans.par1.old <- trans.par1.new
trans.par2.old <- trans.par2.new
pii.old<-pii.new
A.old<-A.new
f0.old<-f0.new
f1.old<-f1.new

## updating the weights and probabilities of hidden states

forwardbackward.res <- forwardbackward1.kernel(x, pii.old, A.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
dgamma <- forwardbackward.res$ts
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

tmp.trans.update <- try(update_trans.prob.nhmm(Z, dgamma, gamma, trans.par1.old,
trans.par2.old,iter.conj.grad=10, dist.included=dist.included))

if(length(tmp.trans.update) > 2){

pii.new <- tmp.trans.update$pii
A.new <- tmp.trans.update$A
trans.par1.new <- tmp.trans.update$trans.par1
trans.par2.new <- tmp.trans.update$trans.par2

npar.A <- dim(Z)[2] + 2

q5<-sum(gamma[, 1]*x)
mu0<-q5/sum(gamma[, 1])

q6<-sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0<-sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
```

```
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

if(symmetric == FALSE){
kern.f1 <- density(x,weights=gamma[,2]/sum(gamma[,2]))
f1.new <- approx(kern.f1$x, kern.f1$y, x, rule = 2, ties="ordered")$y
}

if(symmetric == TRUE){
kern.f1 <- density(c(x,2*f0.new[1]-x),
weights=c(gamma[,2],gamma[,2])/sum(c(gamma[,2],gamma[,2])))
f1.new <- approx(kern.f1$x, kern.f1$y, x, rule = 2, ties="ordered")$y
}

logL.iter <- c(logL.iter,-sum(log(c0)))

df1<-abs(trans.par2.old[-1]-trans.par2.new[-1])
df2<-abs(f1.old-f1.new)
df3 <- abs(logL.iter[niter+1] - logL.iter[niter])

diff<-max(df1, df2, df3)

}

if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

lfdr<-gamma[, 1]
if (converged) {
  logL<--sum(log(c0))
if (nulltype > 0) {
BIC <- logL - (3 + npar.A + 2 - 2)*log(NUM)/2
} else {
BIC <- logL - (3 + npar.A - 2)*log(NUM)/2
}

em.var<-list(pii=pii.new, A=A.new, f0=f0.new, f1=kern.f1, LIS=lfdr, logL=logL, BIC=BIC,
ni=niter, trans.par2 = trans.par2.new, converged=converged,logL.iter=logL.iter[-1])
} else {
BIC<- logL<- (-Inf)
em.var<-list(pii=pii.old, A=A.old, f0=f0.old, f1=kern.f1, LIS=lfdr, logL=logL, BIC= BIC,
ni=niter, trans.par2 = trans.par2.new, converged=converged,logL.iter=logL.iter[-1])
}

}
```

```
if(alttype == 'mixnormal'){

########
# L=1
########

if (L==1)
{

tmp.trans.update <- 0
diff<-1
logL.iter <- 0

while(diff>ptol && niter<maxiter)
{

### control for error in update_trans.prob.nhmm ###

if(length(tmp.trans.update) == 1){

diff<-1
logL.iter <- 0
niter <- 0

trans.par1.new <- c(0,0,0,0)
trans.par2.new <- rnorm(3+dim(Z)[2])
trans.par2.new[4] <- abs(trans.par2.new[4])

tmp.trans.prob <- compute.A.nhmm (Z, trans.par1.new, trans.par2.new,
dist.included=dist.included)
pii.new <- tmp.trans.prob$pii
A.new <- tmp.trans.prob$A

f0.new<-c(0, 1)
locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}
f1.new <- c(4, 1)
}

### The E-M Algorithm

niter<-niter+1

trans.par1.old <- trans.par1.new
```

```
trans.par2.old <- trans.par2.new
pii.old<-pii.new
A.old<-A.new
f0.old<-f0.new
f1.old<-f1.new

## updating the weights and probabilities of hidden states

forwardbackward.res <- forwardbackward1(x, pii.old, A.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
dgamma <- forwardbackward.res$ts
c0 <- forwardbackward.res$rescale

## updating the parameter estimates

tmp.trans.update <- try(update_trans.prob.nhmm(Z, dgamma, gamma, trans.par1.old,
trans.par2.old,iter.conj.grad=10,dist.included=dist.included))

if(length(tmp.trans.update) > 2){

pii.new <- tmp.trans.update$pii
A.new <- tmp.trans.update$A
trans.par1.new <- tmp.trans.update$trans.par1
trans.par2.new <- tmp.trans.update$trans.par2

npar.A <- dim(Z)[2] + 2

q5<-sum(gamma[, 1]*x)
mu0<-q5/sum(gamma[, 1])

q6<-sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0<-sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

q1<-sum(gamma[, 2])
q2<-sum(gamma[, 2]*x)
mu1<-q2/q1
q3<-sum(gamma[, 2]*(x-mu1)*(x-mu1))
sd1<-sqrt(q3/q1)
f1.new<-c(mu1, sd1)
```

```
logL.iter <- c(logL.iter,-sum(log(c0)))

df1<-abs(trans.par2.old[-1]-trans.par2.new[-1])
df2<-abs(f1.old-f1.new)
df3 <- abs(logL.iter[niter+1] - logL.iter[niter])

diff<-max(df1, df2, df3)

}

if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

lfdr<-gamma[, 1]
if (converged) {
  logL<--sum(log(c0))
if (nulltype > 0) {
BIC <- logL - (3*L + npar.A + 2)*log(NUM)/2
} else {
BIC <- logL - (3*L + npar.A)*log(NUM)/2
}

em.var<-list(pii=pii.new, A=A.new, f0=f0.new, f1=f1.new, LIS=lfdr, logL=logL, BIC=BIC,
ni=niter, trans.par2 = trans.par2.new, converged=converged,logL.iter=logL.iter[-1])
} else {
BIC<- logL<- (-Inf)
em.var<-list(pii=pii.old, A=A.old, f0=f0.old, f1=f1.old, LIS=lfdr, logL=logL, BIC= BIC,
ni=niter, trans.par2 = trans.par2.new, converged=converged,logL.iter=logL.iter[-1])
}

}

#######
# L>1
#######

else if (L>1)
{

tmp.trans.update <- 0
diff<-1
logL.iter <- 0

while(diff>ptol && niter<maxiter)
{

### control for error in update_trans.prob.nhmm ###

if(length(tmp.trans.update) == 1){
```

```
diff<-1
logL.iter <- 0
niter <- 0

trans.par1.new <- c(0,0,0,0)
trans.par2.new <- rnorm(3+dim(Z)[2])
trans.par2.new[4] <- abs(trans.par2.new[4])

tmp.trans.prob <- compute.A.nhmm (Z, trans.par1.new, trans.par2.new,dist.included=dist.included)
pii.new <- tmp.trans.prob$pii
A.new <- tmp.trans.prob$A

pc.new<-rep(1, L)/L
mus<-seq(from=-1, by=1.5, length=L)
sds<-rep(1, L)
f0.new<-c(0, 1)
locfdr_p0 <- locfdr(x,plot=0)
if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}
f1.new<-cbind(mus, sds)

}

### The E-M Algorithm

niter<-niter+1

trans.par1.old <- trans.par1.new
trans.par2.old <- trans.par2.new
pii.old<-pii.new
A.old<-A.new
pc.old <- pc.new
f0.old<-f0.new
f1.old<-f1.new

## updating the weights and probabilities of hidden states

forwardbackward.res <- forwardbackward(x, pii.old, A.old, pc.old, f0.old, f1.old)

gamma <- forwardbackward.res$pr
dgamma <- forwardbackward.res$ts
omega <- forwardbackward.res$wt
c0 <- forwardbackward.res$rescale

## updating the parameter estimates
```

```
tmp.trans.update <- try(update_trans.prob.nhmm(Z, dgamma, gamma, trans.par1.old,
trans.par2.old,iter.conj.grad=10,dist.included=dist.included))

if(length(tmp.trans.update) > 2){

pii.new <- tmp.trans.update$pii
A.new <- tmp.trans.update$A
trans.par1.new <- tmp.trans.update$trans.par1
trans.par2.new <- tmp.trans.update$trans.par2

npar.A <- dim(Z)[2] + 2

q5<-sum(gamma[, 1]*x)
mu0<-q5/sum(gamma[, 1])

q6<-sum(gamma[, 1]*(x-mu0)*(x-mu0))
sd0<-sqrt(q6/sum(gamma[, 1]))

f0.new<-c(mu0, sd0)

if(nulltype == 0){
f0.new <- c(0,1)
}
if(nulltype == 1){
f0.new <- c(locfdr_p0$fp0[3,1],locfdr_p0$fp0[3,2])
}
if(nulltype == 2){
f0.new <- c(locfdr_p0$fp0[5,1],locfdr_p0$fp0[5,2])
}

mus<-1:L
sds<-1:L

for (c in 1:L)
{

 q1<-sum(omega[, c])
 q2<-sum(gamma[, 2])
 pc.new[c]<-q1/q2

 q3<-sum(omega[, c]*x)
 mus[c]<-q3/q1

 q4<-sum(omega[, c]*(x-mus[c])*(x-mus[c]))
 sds[c]<-sqrt(q4/q1)

}
f1.new<-cbind(mus, sds)

logL.iter <- c(logL.iter,-sum(log(c0)))
```

```
df1<-abs(trans.par2.old[-1]-trans.par2.new[-1])
df2<-abs(f1.old-f1.new)
df3 <- abs(logL.iter[niter+1] - logL.iter[niter])

diff<-max(df1, df2, df3)

}

if (is.na(diff)) {
   converged=FALSE;
   break;
}

}

 lfdr<-gamma[, 1]
if (converged) {
 logL <- -sum(log(c0))
if (nulltype>0) {
BIC <- logL - (3*L + npar.A + 2)*log(NUM)/2
} else {
BIC <- logL - (3*L + npar.A)*log(NUM)/2
}
em.var<-list(pii=pii.new, A=A.new, pc=pc.new, f0=f0.new, f1=f1.new, LIS=lfdr, logL=logL,
BIC=BIC, ni=niter, trans.par2 = trans.par2.new, converged=converged,
logL.iter=logL.iter[-1])
} else {
 logL<- (-Inf)
 BIC<- logL<- (-Inf)
 em.var <- list(pii=pii.old, A=A.old, pc=pc.old, f0=f0.old, f1=f1.old, LIS=lfdr, logL=logL,
 BIC=BIC, ni=niter, trans.par2 = trans.par2.new, converged=converged,
 logL.iter=logL.iter[-1])
}

}
}

return (em.var)
}
```

`fdr.nhmm` *FDR control based for multiple hypothesis tests under dependence*

Description

Main function which fits the NHMM and HMM based fdr control. It offers numerous options for FDR control as described in Kuan et al. (2011).

Usage

```
fdr.nhmm(x, Z = NULL, dist = NULL, log.transform.dist = TRUE, alttype = "kernel", L = 2,
maxiter = 1000, nulltype = 0, modeltype = "NHMM", symmetric = FALSE,epsilon =1e-04)
```

Arguments

x	A vector of the observed data. It is assumed that the data has been ordered by time or position.
Z	A matrix of covariates EXCLUDING spacing/distance between probes. Each column corresponds to one covariate. If there is only one covariate, define Z to be a matrix with one column. If fitting a homogeneous HMM, i.e., no covariate is required, set Z = NULL.
dist	A vector of spacing/distance between probes. If fitting a homogeneous HMM, i.e., no covariate is required, set dist = NULL.
log.transform.dist	Logical value. TRUE is log2(Distance+2) is to be applied to the dist covariate. Recommended for numerical stability.
alttype	Type of estimating for alternative hypothesis f1(x). Available choices are "mixnormal" for gaussian mixtures or "kernel" for non-parametric kernel density estimates.
L	Number of mixture component for alttype = "mixnormal". If alttype = "kernel", this is irrelevant.
maxiter	Maximum iterations in the EM algorithm to speed up computation.
nulltype	Type of null hypothesis assumed in estimating f0(z), for use in the fdr calculations. 0 is the theoretical null N(0,1), 1 is maximum likelihood estimation, 2 is central matching estimation. This method is imported from R package locfdr.\ NOTE: Recommended to use nulltype = 0, i.e., theoretical null to avoid double correction of correlation structure.
modeltype	Types of dependence structure. Available choices are "Indep", "HMM" or "NHMM". See details.
symmetric	Logical value. TRUE if the alternative hypothesis f1(x) is assumed to be symmetrical. Note that this option is only available for alttype = "kernel".
epsilon	Convergence control in the EM algorithm.

Details

modeltype = "Indep" assumes the tests are independent. modeltype = "HMM" assumes the dependence structure follows a homogeneous HMM based on Sun and Cai (2009).

Value

LIS	Local index of significance (analog of p-values). This will be used for FDR control in LIS.adjust.
BIC	Bayesian information criterion (BIC).
pii	Initial probabilities.
A	Transition probability matrix.

f0	Null hypothesis.
f1	Alternative hypothesis.
logL	Log likelihood.
trans.par2	Transition parameters for State 1.

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

See Also

LIS.adjust

Examples

```
library(NHMMfdr)

################
# Simulate data
################

### simulate covariate and transition prob
NUM1 <- 1000
Z <- rnorm(NUM1)
Z <- matrix(Z,ncol=1)

Z <- apply(Z,2,scale)
trans.par1.true <- c(0,0,0,0)

trans.par2.true <- rnorm(3+dim(Z)[2])

print(trans.par2.true[-1])

A.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
dist.included=FALSE)$A
pii.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
  dist.included=FALSE)$pii

### the null distribution
```

```
f0 <- c(0, 1)

### the alternative distribution
f1 <- c(3, 1)

### the NHMM data

simdat <- simdata.nhmm(NUM1, pii.true, A.true, f0, 1, f1)

### the observed values
x <- simdat$o

### the unobserved true states
theta1 <- simdat$s

################
# Model fitting
################

fit.nhmm <- fdr.nhmm(x, Z, dist = NULL, log.transform.dist = FALSE,
alttype ='mixnormal', L=1, maxiter = 100, nulltype = 0, modeltype = 'NHMM', epsilon=1e-4)

### checking estimated parameters
print(fit.nhmm$trans.par2[-1])

################
# Adjust LIS
################

LIS.adjust <- LIS.adjust(fit.nhmm$LIS, fdr = 0.1, adjust = TRUE)

### tests which are statistically significant

sig.test <- which(LIS.adjust$States == 1)
length(sig.test)
```

`find.nu` *Intermediate function*

Description

Intermediate function

Usage

```
find.nu(Z, dgamma, gamma, delta1, delta2, trans.par1, trans.par2,dist.included = TRUE)
```

Arguments

Z

dgamma

gamma

delta1

delta2

trans.par1

trans.par2

dist.included

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(Z, dgamma, gamma, delta1, delta2, trans.par1, trans.par2, dist.included=TRUE){

p <- dim(Z)[2]
# delta1 = (lambda1_H, sigma11_H, sigma21_H, rho1_H)
delta_H <- array(0,c(2,2,c(dim(Z)[1]-1)))
delta1_H0 <- delta1[1] + sum(delta1[-c(1:3)]*Z[1,])
delta2_H0 <- delta2[1] + sum(delta2[-c(1:3)]*Z[1,])

tmp11 <- t(delta1[-c(1:3)]*t(Z[-1,]))
tmp21 <- t(delta1[-c(1:3)]*t(Z[-1,]))
tmp12 <- t(delta2[-c(1:3)]*t(Z[-1,]))
tmp22 <- t(delta2[-c(1:3)]*t(Z[-1,]))

        if(dist.included==TRUE){
          delta1_H0 <- delta1[1] + sum(delta1[-c(1:4)]*Z[1,-1])
          delta2_H0 <- delta2[1] + sum(delta2[-c(1:4)]*Z[1,-1])
```

```
          #tmp11 <- t(c(delta1[4],delta1[-c(1:4)])*t(Z[-1,]))
          tmp21 <- t(c(-delta1[4],delta1[-c(1:4)])*t(Z[-1,]))
          #tmp12 <- t(c(delta2[4],delta2[-c(1:4)])*t(Z[-1,]))
          tmp22 <- t(c(-delta2[4],delta2[-c(1:4)])*t(Z[-1,]))
        }

denom11 = denom21 = denom12 = denom22 <- rep(0,dim(Z)[1]-1)

for(i in 1:p){
denom11 <- denom11 + tmp11[,i]
denom21 <- denom21 + tmp21[,i]
denom12 <- denom12 + tmp12[,i]
denom22 <- denom22 + tmp22[,i]
}

delta_H[1,1,] <- delta1[2] + denom11
delta_H[2,1,] <- delta1[3] + denom21
delta_H[1,2,] <- delta2[2] + denom12
delta_H[2,2,] <- delta2[3] + denom22


nu.new <- 0
ptol <- 1
iter <- 0

while(ptol > 1e-6){
iter <- iter + 1

trans.par1.new <- trans.par1 + nu.new*delta1
trans.par2.new <- trans.par2 + nu.new*delta2

tmp.trans.prob <- compute.A.nhmm (Z, trans.par1.new, trans.par2.new,
dist.included=dist.included)
pii <- tmp.trans.prob$pii
A <- tmp.trans.prob$A

f <- sum(c(delta1_H0,delta2_H0)*(gamma[1,] - pii))
for(i in 1:2){
for(j in 1:2){
f <- f + sum(delta_H[i,j,]*(dgamma[i,j,] - gamma[-dim(gamma)[1],i]*A[i,j,]))
}
}

     fprime <- -sum(c(delta1_H0,delta2_H0)^2*pii*(1 - pii))
for(i in 1:2){
for(j in 1:2){
fprime <- fprime -
sum(delta_H[i,j,]^2*gamma[-dim(gamma)[1],i]*A[i,j,]*(1 - A[i,j,]))
}
}
nu.old <- nu.new
nu.new <- nu.old - f/fprime
ptol <- abs(f)
```

```
if(is.na(ptol)||iter > 100){
nu.new <- NaN
break
}
}
trans.par1.new <- trans.par1 + nu.new*delta1
trans.par2.new <- trans.par2 + nu.new*delta2

return(list(nu=nu.new,trans.par1=trans.par1.new,trans.par2= trans.par2.new))
}
```

`forwardbackward` *Intermediate function*

Description

Intermediate function

Usage

```
forwardbackward(x, pii, A, pc, f0, f1)
```

Arguments

`x`

`pii`

`A`

`pc`

`f0`

`f1`

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, pii, A, pc, f0, f1)
{

NUM<-length(x)
L<-length(pc)


f0x<-dnorm(x, f0[1], f0[2])
f1x<-rep(0, NUM)
for (c in 1:L)
{
  f1x<-f1x+pc[c]*dnorm(x, f1[c, 1], f1[c, 2])
}


alpha<-matrix(rep(0, NUM*2), NUM, 2, byrow=TRUE)
c0<-rep(0, NUM)

alpha[1, 1]<-pii[1]*f0x[1]
alpha[1, 2]<-pii[2]*f1x[1]
c0[1]<-1/sum(alpha[1, ])
alpha[1, ]<-c0[1]*alpha[1, ]

alpha.tmp <- .C('calAlpha',alpha=as.numeric(alpha),c0=as.numeric(c0),as.numeric(A),
as.numeric(f0x),as.numeric(f1x),as.integer(NUM))

alpha <- alpha.tmp$alpha
dim(alpha) <- c(NUM,2)

c0 <- alpha.tmp$c0

beta<-matrix(rep(0, NUM*2), NUM, 2, byrow=TRUE)

beta[NUM, 1]<-c0[NUM]
beta[NUM, 2]<-c0[NUM]

beta.tmp <- .C('calBeta',beta=as.numeric(beta),as.numeric(c0),as.numeric(A),
as.numeric(f0x),as.numeric(f1x),as.integer(NUM))

beta <- beta.tmp$beta
dim(beta) <- c(NUM,2)

lfdr<-rep(0, NUM)

lfdr.tmp <- .C('calLfdr',as.numeric(alpha),as.numeric(beta),lfdr=as.numeric(lfdr),
as.integer(NUM))
```

```
lfdr <- lfdr.tmp$lfdr

gamma<-matrix(1:(NUM*2), NUM, 2, byrow=TRUE)
gamma[NUM, ]<-c(lfdr[NUM], 1-lfdr[NUM])
dgamma<-array(rep(0, (NUM-1)*4), c(2, 2, (NUM-1)))

gamma.tmp <- .C('calGamma',as.numeric(alpha),as.numeric(beta),as.numeric(A),as.numeric(f0x),
as.numeric(f1x),gamma=as.numeric(gamma),dgamma=as.numeric(dgamma),as.integer(NUM))

gamma <- gamma.tmp$gamma
dgamma <- gamma.tmp$dgamma
dim(gamma) <- c(NUM,2)
dim(dgamma) <- c(2, 2, (NUM-1))

omega<-matrix(rep(0, NUM*L), NUM, L, byrow=TRUE)

 for (c in 1:L)
  {
    f1c<-dnorm(x, f1[c, 1], f1[c, 2])
    omega[, c]<-gamma[, 2]*pc[c]*f1c/f1x
  }


forwardbackward.var<-list(bw=alpha, fw=beta, lf=lfdr, pr=gamma, ts=dgamma, wt=omega,
rescale=c0)
return(forwardbackward.var)

}
```

---

forwardbackward1 *Intermediate function*

Description

Intermediate function

Usage

```
forwardbackward1(x, pii, A, f0, f1)
```

Arguments

x

pii

A

f0

f1

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, pii, A, f0, f1)
{

## Initialize

NUM<-length(x)

## Densities

f0x<-dnorm(x, f0[1], f0[2])
f1x<-dnorm(x, f1[1], f1[2])

## the backward-forward procedure

alpha<-matrix(rep(0, NUM*2), NUM, 2, byrow=TRUE)

c0<-rep(0, NUM)

alpha[1, 1]<-pii[1]*f0x[1]
alpha[1, 2]<-pii[2]*f1x[1]

c0[1]<-1/sum(alpha[1, ])
alpha[1, ]<-c0[1]*alpha[1, ]

alpha.tmp <- .C('calAlpha',alpha = as.numeric(alpha),c0=as.numeric(c0),as.numeric(A),
as.numeric(f0x),as.numeric(f1x),as.integer(NUM))

alpha <- alpha.tmp$alpha
dim(alpha) <- c(NUM,2)

c0 <- alpha.tmp$c0
```

```
beta<-matrix(rep(0, NUM*2), NUM, 2, byrow=TRUE)

beta[NUM, 1]<-c0[NUM]
beta[NUM, 2]<-c0[NUM]

beta.tmp <- .C('calBeta',beta=as.numeric(beta),as.numeric(c0),as.numeric(A),
as.numeric(f0x),as.numeric(f1x),as.integer(NUM))

beta <- beta.tmp$beta
dim(beta) <- c(NUM,2)

lfdr<-rep(0, NUM)

lfdr.tmp <- .C('calLfdr',as.numeric(alpha),as.numeric(beta),lfdr = as.numeric(lfdr),
as.integer(NUM))

lfdr <- lfdr.tmp$lfdr

gamma<-matrix(1:(NUM*2), NUM, 2, byrow=TRUE)
gamma[NUM, ]<-c(lfdr[NUM], 1-lfdr[NUM])
dgamma<-array(rep(0, (NUM-1)*4), c(2, 2, (NUM-1)))

gamma.tmp <- .C('calGamma',as.numeric(alpha),as.numeric(beta),as.numeric(A),as.numeric(f0x),
as.numeric(f1x),gamma=as.numeric(gamma),dgamma=as.numeric(dgamma),as.integer(NUM))

gamma <- gamma.tmp$gamma
dgamma <- gamma.tmp$dgamma
dim(gamma) <- c(NUM,2)
dim(dgamma) <- c(2, 2, (NUM-1))

forwardbackward.var<-list(bw=alpha, fw=beta, lf=lfdr, pr=gamma, ts=dgamma, rescale=c0)
return(forwardbackward.var)

}
```

`forwardbackward1.indep` *Intermediate function*

Description

Intermediate function

Usage

```
forwardbackward1.indep(x, ptheta, f0, f1)
```

Arguments

x

ptheta

f0

f1

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, ptheta, f0, f1)
{


## Initialize

NUM<-length(x)

## Densities

f0x<-dnorm(x, f0[1], f0[2])
f1x<-dnorm(x, f1[1], f1[2])

gamma<-matrix(1:(NUM*2), NUM, 2, byrow=TRUE)

gamma[,1] <- ptheta[1]*f0x/(ptheta[1]*f0x + ptheta[2]*f1x)
gamma[,2] <- 1 - gamma[,1]

lfdr <- gamma[,1]

c0 <- f0x*ptheta[1] + f1x*ptheta[2]
```

```
forwardbackward.var<-list(lf=lfdr, pr=gamma,rescale=c0)
return(forwardbackward.var)

}
```

`forwardbackward1.indep.kernel` *Intermediate function*

Description

Intermediate function

Usage

```
forwardbackward1.indep.kernel(x, ptheta, f0, f1x)
```

Arguments

`x`

`ptheta`

`f0`

`f1x`

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, ptheta, f0, f1x)
{
```

```
## Initialize

NUM<-length(x)

## Densities

f0x<-dnorm(x, f0[1], f0[2])

gamma<-matrix(1:(NUM*2), NUM, 2, byrow=TRUE)

gamma[,1] <- ptheta[1]*f0x/(ptheta[1]*f0x + ptheta[2]*f1x)
gamma[,2] <- 1 - gamma[,1]

lfdr <- gamma[,1]

c0 <- f0x*ptheta[1] + f1x*ptheta[2]

forwardbackward.var<-list(lf=lfdr, pr=gamma,rescale=c0)
return(forwardbackward.var)

}
```

`forwardbackward1.kernel` *Intermediate function*

Description

Intermediate function

Usage

```
forwardbackward1.kernel(x, pii, A, f0, f1x)
```

Arguments

`x`

`pii`

`A`

`f0`

`f1x`

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x, pii, A, f0, f1x)
{

## Initialize

NUM<-length(x)

## Densities

f0x<-dnorm(x, f0[1], f0[2])

## the backward-forward procedure

alpha<-matrix(rep(0, NUM*2), NUM, 2, byrow=TRUE)

c0<-rep(0, NUM)

alpha[1, 1]<-pii[1]*f0x[1]
alpha[1, 2]<-pii[2]*f1x[1]

c0[1]<-1/sum(alpha[1, ])
alpha[1, ]<-c0[1]*alpha[1, ]

alpha.tmp <- .C('calAlpha',alpha = as.numeric(alpha),c0=as.numeric(c0),as.numeric(A),
as.numeric(f0x),as.numeric(f1x),as.integer(NUM))

alpha <- alpha.tmp$alpha
dim(alpha) <- c(NUM,2)

c0 <- alpha.tmp$c0

beta<-matrix(rep(0, NUM*2), NUM, 2, byrow=TRUE)

beta[NUM, 1]<-c0[NUM]
beta[NUM, 2]<-c0[NUM]
```

```
beta.tmp <- .C('calBeta',beta=as.numeric(beta),as.numeric(c0),as.numeric(A),
as.numeric(f0x),as.numeric(f1x),as.integer(NUM))

beta <- beta.tmp$beta
dim(beta) <- c(NUM,2)

lfdr<-rep(0, NUM)

lfdr.tmp <- .C('calLfdr',as.numeric(alpha),as.numeric(beta),lfdr = as.numeric(lfdr),
as.integer(NUM))

lfdr <- lfdr.tmp$lfdr

gamma<-matrix(1:(NUM*2), NUM, 2, byrow=TRUE)
gamma[NUM, ]<-c(lfdr[NUM], 1-lfdr[NUM])
dgamma<-array(rep(0, (NUM-1)*4), c(2, 2, (NUM-1)))

gamma.tmp <- .C('calGamma',as.numeric(alpha),as.numeric(beta),as.numeric(A),as.numeric(f0x),
as.numeric(f1x),gamma=as.numeric(gamma),dgamma=as.numeric(dgamma),as.integer(NUM))

gamma <- gamma.tmp$gamma
dgamma <- gamma.tmp$dgamma
dim(gamma) <- c(NUM,2)
dim(dgamma) <- c(2, 2, (NUM-1))

forwardbackward.var<-list(bw=alpha, fw=beta, lf=lfdr, pr=gamma, ts=dgamma, rescale=c0)
return(forwardbackward.var)

}
```

`LIS.adjust` *Determine statistically significant tests at a user-specified FDR level.*

Description

Compute adjusted local index of significance (LIS) and determine the tests which are statistically significant at a user-specified FDR level.

Usage

```
LIS.adjust(lis, fdr = 0.001, adjust = TRUE)
```

Arguments

`lis`	LIS scores from `fdr.nhmm`.
`fdr`	Desirable FDR level. Usually 0.05.
`adjust`	Logical value. `TRUE` if adjusted LIS is to be computed. See `details`.

Details

If `adjust = TRUE`, the adjusted LIS will be computed. This will be useful if the user wants to control at different FDR level. The significant tests corresponds to those adjusted LIS which are less than the FDR level.

Value

`States`	A binary valued vector. Takes value 1 if the test or observation is declared to be significant.
`aLIS`	Adjusted LIS.

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

See Also

`fdr.nhmm`

Examples

```
library(NHMMfdr)

################
# Simulate data
################

### simulate covariate and transition prob
NUM1 <- 1000
Z <- rnorm(NUM1)
Z <- matrix(Z,ncol=1)

Z <- apply(Z,2,scale)
trans.par1.true <- c(0,0,0,0)

trans.par2.true <- rnorm(3+dim(Z)[2])

print(trans.par2.true[-1])

A.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
dist.included=FALSE)$A
pii.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true,
  dist.included=FALSE)$pii
```

```
### the null distribution
f0 <- c(0, 1)

### the alternative distribution
f1 <- c(3, 1)

### the NHMM data

simdat <- simdata.nhmm(NUM1, pii.true, A.true, f0, 1, f1)

### the observed values
x <- simdat$o

### the unobserved true states
theta1 <- simdat$s

#################
# Model fitting
#################

fit.nhmm <- fdr.nhmm(x, Z, dist = NULL, log.transform.dist = FALSE,
alttype ='mixnormal', L=1, maxiter = 100, nulltype = 0, modeltype = 'NHMM', epsilon=1e-4)

### checking estimated parameters
print(fit.nhmm$trans.par2[-1])

#################
# Adjust LIS
#################

LIS.adjust <- LIS.adjust(fit.nhmm$LIS, fdr = 0.1, adjust = TRUE)

### tests which are statistically significant

sig.test <- which(LIS.adjust$States == 1)
length(sig.test)
```

`simdata.nhmm` *Simulate from a non-homogeneous hidden Markov Model (NHMM).*

Description

Function to simulate observations from a NHMM.

Usage

```
simdata.nhmm(NUM, pii, A, f0, pc, f1)
```

Arguments

NUM	Number of observations.
pii	Initial probabilities.
A	Transition probability matrix.
f0	Null distribution.
pc	Mixture proportion for alternative distribution.
f1	Alternative distribution.

Value

s	Simulated hidden states.
o	Simulated observations.

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2011). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

See Also

fdr.nhmm, LIS.adjust

Examples

```
library(NHMMfdr)

################
# Simulate data
################

### simulate covariate and transition prob
NUM1 <- 1000
Z <- rnorm(NUM1)
Z <- matrix(Z,ncol=1)

Z <- apply(Z,2,scale)
trans.par1.true <- c(0,0,0,0)

trans.par2.true <- rnorm(3+dim(Z)[2])

print(trans.par2.true[-1])

A.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true)$A
```

```
pii.true <- compute.A.nhmm(Z, trans.par1.true, trans.par2.true)$pii

### the null distribution
f0 <- c(0, 1)

### the alternative distribution
f1 <- c(3, 1)

### the NHMM data

simdat <- simdata.nhmm(NUM1, pii.true, A.true, f0, 1, f1)

### the observed values
x <- simdat$o

### the unobserved true states
theta1 <- simdat$s

#################
# Model fitting
#################

fit.nhmm <- fdr.nhmm(x, Z, alttype ='mixnormal', L=1, maxiter = 100, nulltype = 0,
modeltype = 'NHMM', epsilon=1e-4)

### checking estimated parameters
print(fit.nhmm$trans.par2[-1])

#################
# Adjust LIS
#################

LIS.adjust <- LIS.adjust(fit.nhmm$LIS, fdr = 0.1, adjust = TRUE)

### tests which are statistically significant

sig.test <- which(LIS.adjust$States == 1)
length(sig.test)
```

`simdata1.nhmm` *Intermediate function*

Description

Intermediate function

Usage

```
simdata1.nhmm(NUM, pii, A, f0, f1)
```

Arguments

NUM

pii

A

f0

f1

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(NUM, pii, A, f0, f1)
{


theta<-rep(0, NUM)
x<-rep(0, NUM)

## generating the states
 # initial state
theta[1]<-rbinom(1, 1, pii[2])
 # other states
for (i in 2:NUM)
{
  if (theta[i-1]==0)
     theta[i]<-rbinom(1, 1, A[1, 2, i-1])
  else
     theta[i]<-rbinom(1, 1, A[2, 2, i-1])
}

## generating the observations
for (i in 1:NUM)
```

```
{
  if (theta[i]==0)
  {
    x[i]<-rnorm(1, mean=f0[1], sd=f0[2])
  }
  else
  {
    x[i]<-rnorm(1, mean=f1[1], sd=f1[2])
  }
}
data<-list(s=theta, o=x)
return (data)

}
```

---

update_delta2 *Intermediate function*

Description

Intermediate function

Usage

```
update_delta2(Z, dgamma, gamma, delta.old, gradient.new, gradient.old)
```

Arguments

Z

dgamma

gamma

delta.old

gradient.new

gradient.old

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(Z, dgamma, gamma, delta.old, gradient.new, gradient.old){

num.fac <- sum((gradient.new - gradient.old)*gradient.new)
denom.fac <- sum(gradient.old^2)
fac <- num.fac/denom.fac
if(fac<0) fac <- 0
delta2 <- -gradient.new + fac*delta.old
return(delta2)
}
```

---

update_trans.prob.nhmm

Intermediate function

Description

Intermediate function

Usage

```
update_trans.prob.nhmm(Z, dgamma, gamma, trans.par1, trans.par2, iter.conj.grad = 10,
dist.included = TRUE)
```

Arguments

Z
dgamma
gamma
trans.par1
trans.par2
iter.conj.grad
dist.included

Author(s)

Pei Fen Kuan

References

P.F. Kuan and D.Y. Chiang (2012). Integrating Prior Knowledge in Multiple Testing Under Dependence with Applications in Detecting Differential DNA Methylation. Biometrics, doi: 10.1111/j.1541-0420.2011.01730.x.

B. Efron (2004). Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. Journal of the American Stat. Assoc. 99, 96-104.

W. Sun and T. Cai (2009). Large-scale multiple testing under dependence. J. R.Stat. Soc B. 71, 393-424.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(Z, dgamma, gamma, trans.par1, trans.par2,iter.conj.grad=10, dist.included=TRUE){

trans.par1.old <- trans.par1
trans.par2.old <- trans.par2

gradient.old <- compute.gradient(Z, dgamma, gamma, trans.par1.old, trans.par2.old,
dist.included=dist.included)
delta2.old <- -gradient.old

n.conj.grad = 0
while(n.conj.grad <= iter.conj.grad){

tmp <- try(find.nu(Z, dgamma, gamma, c(0,0,0,0),delta2.old,trans.par1.old, trans.par2.old,
dist.included=dist.included))

if(is.na(tmp$nu)){
break
}
trans.par1.new <- tmp$trans.par1
trans.par2.new <- tmp$trans.par2

gradient.new <- compute.gradient(Z, dgamma, gamma,trans.par1.new, trans.par2.new,
dist.included=dist.included)
delta2.new <- update_delta2(Z, dgamma, gamma, delta2.old, gradient.new, gradient.old)

trans.par1.old <- trans.par1.new
trans.par2.old <- trans.par2.new
        if(dist.included==TRUE&trans.par2.new[4]<0)trans.par2.new[4] <- 0.5
gradient.old <- gradient.new
delta2.old <- delta2.new
n.conj.grad <- n.conj.grad + 1

}

if(!is.na(tmp$nu)){
```

```
tmp.trans.prob <- compute.A.nhmm(Z, trans.par1.new, trans.par2.new,
dist.included=dist.included)
pii.new <- tmp.trans.prob$pii
A.new <- tmp.trans.prob$A
return(list(A = A.new, pii = pii.new, trans.par1 = trans.par1.new,
trans.par2 = trans.par2.new))
}else return(1)
}
```

Index