

Example workflow

We illustrate a typical work flow using mvama with a small generated data set. The reader is encouraged to copy blocks of commands into R and explore the objects produced in conjunction with the mvama library help files.

1. Get the data – we want the expression data matrix X, the design matrix D and the contrast matrix C

```
library(Matrix)
library(mvama)
set.seed(12) # just for reproducibility of what follows
# small 3 group problem with 5 variables and 150 samples
# generate mean matrix
a<-as.numeric(gl(3,50,150))
# The three groups are observations 1-50, 51-100, and 101-150
b<-c(0,1,0,1,1)
M<-outer(a,b,"*") # variables 2,4 and 5 have nonzero and unequal group means
# design matrix
D<-matrix(0,nrow=150,ncol=3)
D[1:50,1]<-1
D[51:100,2]<-1
D[101:150,3]<-1
# contrast matrix for testing equality of group2 and 1 and group 3 and group 1 means
C<-matrix(c(1,-1,0,1,0,-1)/2^0.5,ncol=2)
# Get estimated sigma inverse - this will typically be obtained by using the package
# sparse.inv.cov but we make one up here for illustrative purposes.
sinv<-diag(5)/2
sinv[1,2]=sinv[2,3]=sinv[3,4]=sinv[4,5]=0.1
sinv<-sinv+t(sinv)
sinv<-as(sinv,"dsCMatrix")

# generate data with this mean and (inverse) covariance structure
res<-Simulate(sigma.inv=sinv,M=M,nsamp=150,nsim=1,seed=123)
X<-res$data[[1]]
# define "gene" names
gn<-c("a","b","c","d","e")
# for graphs later
```

```
colnames(sinv)<-gn
```

```
# If we know the pattern of zeroes a relatively fast estimate of sigma.inv can be  
# computed from the data by using
```

```
a<-(sinv!=0)*1 # known pattern say  
a<-as(a,"dsCMatrix")  
R<-lm(X~D)$residuals  
sinv1<-sigma.inv.approx(R,a,symmetric=TRUE)  
# symmetric=FALSE might be useful also
```

```
# In this example we use the theoretical known value of sigma inverse.  
# To do both steps of finding the zero pattern and estimating the inverse covariance  
# matrix see the R library sparse.inv.cov available at  
# www.bioinformatics.csiro.au/sparse.inv.cov/index.shtml  
#
```

```
# Coming soon - variable transformations to better approximate the multivariate  
normal distribution
```

2. Generate Null distributions by permutation

```
#Either-----  
# If you have the space, time and patience do it all on one processor  
# Here we do 200 permutations  
# For choice of significance levels see the paper, .05 chosen here for illustrative  
# purposes only.  
res<-perms(C,X,D,nperm=200,seed=123)  
#check some components  
length(res$CTB)  
dim(res$CTB[[1]])  
#observed contrast values  
res$ctb0  
# Get summaries of permutation distributions (pval=.05) for later testing  
tmp<-perm.summary(res,sigma.inv=sinv,pval=.05,nval=NULL,double.sided=TRUE,  
tails=FALSE)
```

```
#Or-----  
#Alternatively split the job over two or more processors  
# Below is an example splitting over two processors  
# generate permutation distributions - get quantiles for 200 permutations by  
# combining two sets of 100 (too small really) permutations each  
# Note than nval is computed from the TOTAL number of permutations, and pval is  
# the same for both runs. The seeds must be different for the different runs!!
```

```
# on processor 1 say  
nval<-trunc(200*.05/2+0.5)+2  
res1<-perms(C,X,D,nperm=100,seed=123)  
a1<-perm.summary(res1,sigma.inv=sinv,pval=.05,nval=nval,double.sided=TRUE,  
tails=TRUE)  
save(a1,file="perm1.RData")
```

```

# on processor 2
nval<-trunc(200*.05/2+0.5)+2
res2<-perms(C,X,D,nperm=100,seed=456)
a2<-perm.summary(res2,sigma.inv=sinv,pval=.05,nval=nval,double.sided=TRUE,
                 tails=TRUE)
save(a2,file="perm2.RData")
#
# start R in the directory with perm1.RData and perm2.RData
#
a<-list()
load("perm1.RData")
a[[1]]<-a1
load("perm2.RData")
a[[2]]<-a2
# combine the results
tmp<-combine.quantiles(a)
#-----
# Note - results from the two methods above will not be identical because the set of
# permutations produced is not the same (different seed(s) are used in the second
# method). This is less noticeable with large numbers of permutations.
# In practice the number of permutations needs to be much larger than used in this toy
# example.

```

3. Get observed test statistic values

```
stat<-get.statistics(tmp,sinv)
```

4. Perform significance tests

```
# first do overall tests for contrasts
```

```
to<- significance.tests(stat,tmp, type = c("T"), top = 3, gn)
```

```
# then test for contributions of individual genes
```

```
tc<- significance.tests(stat,tmp, type = c("cT"), top = 3, gn)
```

```
# plot components of T to identify possible interesting connections for contrast 1
```

```
q<-tmp$Tcq[[1]] # quantiles of NULL distribution of components of T
```

```
v<-stat$cT[,1] # observed components of T for contrast 1
```

```
# plot components of T (black) and significance levels (in red)
```

```
plot(q[1,],ty="l",lty=3,col=2,ylim=c(-.1,1.1),ylab="cT value",xlab="variable")
```

```
lines(q[5,],col=2,lty=3)
```

```
lines(v)
```

```
points(v)
```

```
# as expected variables 2, 4 and 5 stand out
```

```
# break down further into differential expression
```

```
tde<- significance.tests(stat,tmp, type = c("de"), top = 3, gn)
```

```
tde[[1]]$topids #ids of top differentially expressed genes for contrast 1
```

```
tde[[1]]$allids # ids of all differentially expressed genes for contrast 1
```

```

# and differential connection
tdc<- significance.tests(stat,tmp, type = c("dc"), top = 3, gn)
tdc[[2]]$topids #ids of top differentially connected genes for contrast 2
tdc[[2]]$allids # ids of all differentially connected genes for contrast 2

# For this artificial example all ‘genes’ are known to be differentially connected.
# Detection depends on the noise level and observed values of the neighbour
# corrected contrasts.

5. Plot graphs of interest
# make graph object and plot – colour with differentially expressed genes for
# contrast 1
k<-tde[[1]]$allids
de<-rep(0,5)
de[k]<-1
g<-make.graph(sinv,gn=gn,, ide=de)
# plot
netplot(g,vertex.label.color="black")
# warning – do not attempt to plot a very large graph

# find neighbourhoods of size one of each of the genes ‘b’ and ‘d’ and plot them
resp<-graph.neighb(g,order=1, c("b","d"))
# plot neighbourhood of gene ‘b’
netplot(resp[[1]])
#plot neighbourhood of gene ‘d’
netplot(resp[[2]])

# do a prediction plot for gene ‘b’
prediction.plot(gene.id="c",sinv,obj=res,contrast=1)

# find clusters of the subgraph restricted to nodes "a","b","d","e".
# for real data more interesting for subgraph defined by differentially expressed genes
resc<-find.clusters(g, c("a","b","d","e"))
resc$size # cluster sizes
resc$cno # number of clusters
#look at results
netplot(g) # plot original graph
netplot(resc$gs) # plot subgraph
netplot(resc$clust.list[[1]]) # plot cluster 1 graph
netplot(resc$clust.list[[2]]) # plot cluster 2 graph

# get paths from ‘a’ to ‘d’,’e’
resg<-get.paths(g,from="a",to=c("d","e"))
# print out paths
resg

```