

Annotated Network Analysis Code

Yale affy exon arrays 102407

The data is from Affymetrix exon arrays; 93 arrays have been collected in all, from four brains and five major brain regions (neocortex, hippocampus, striatum, thalamus, and cerebellum). In addition, neocortical samples have been collected from nine distinct areas, and nearly all samples have also been collected from both left and right hemispheres.

setting the work directories-----

```
setwd("/data/home/fuying/Yale")
```

loading the packages-----

```
#library(exonmap) #read in exon cel files
```

```
library(MASS)
library(gplots)
library(MASS)
library(class)
library(cluster)
library(sma)
library(impute)
library(scatterplot3d)
```

```
source("NetworkFunctions.txt")
source("MO_fxs.txt")
```

```
curr.study="Yale"
```

1. Reading data in & outlier exclusion

```
narrays<-93
dat1=read.csv("4brainsabsolutevalue.csv",sep="," ,header=T)
dim(dat1)
# [1] 17868 110
```

```
[1] "Transcript.ID"      "gene_assignment"  "X18.1"            "X18.3"
[5] "X18.5"              "X18.7"            "X18.9"            "X18.11"
[9] "X18.13"             "X18.15"           "X18.17"           "X18.19"
[13] "X18.21"             "X18.23"           "X18.25"           "X18.27"
[17] "X18.29"             "X18.31"           "X18.33"           "X18.35"
[21] "X18.37"             "X18.39"           "X18.41"           "X18.43"
[25] "X18.45"             "X19.1"            "X19.3"            "X19.5"
[29] "X19.7"              "X19.9"            "X19.11"           "X19.13"
[33] "X19.15"             "X19.17"           "X19.19"           "X19.21"
[37] "X19.23"             "X19.25"           "X19.27"           "X19.29"
[41] "X19.31"             "X19.33"           "X19.35"           "X19.37"
[45] "X19.39"             "X19.41"           "X19.43"           "X19.45"
[49] "X19.47"             "X19.49"           "X19.51"           "X19.53"
[53] "X19.54"             "X19.55"           "X19.56"           "X19.57"
[57] "X19.58"             "X19.59"           "X19.61"           "X21.1"
[61] "X21.3"              "X21.5"            "X21.7"            "X21.9"
[65] "X21.11"             "X21.13"           "X21.15"           "X21.17"
[69] "X21.19"             "X21.21"           "X21.23"           "X21.25"
[73] "X21.27"             "X21.29"           "X21.31"           "X21.33"
[77] "X21.35"             "X21.37"           "X21.39"           "X21.41"
[81] "X21.43"             "X21.45"           "X23.1"            "X23.3"
[85] "X23.5"              "X23.7"            "X23.9"            "X23.11"
[89] "X23.13"             "X23.15"           "X23.17"           "X23.19"
```

```

[93] "X23.21"      "X23.23"      "X23.25"      "X23.27"
[97] "X23.29"      "X23.31"      "X23.33"      "X23.35"
[101] "X23.37"      "X23.39"      "X23.41"      "X23.43"
[105] "X23.45"      "C18"         "C19.2"       "C19"
[109] "C21"         "C23"

```

```

Samples<-read.delim(file="samples.txt", header=T)
labls<-paste(Samples$Region, Samples$SampleName, sep="_")

```

```

colnames(dat1)[3:110]<-labls

```

```

[1] "Transcript.ID"  "gene_assignment" "Nctx_18L_D"
[4] "Nctx_18R_D"     "Nctx_18L_V"     "Nctx_18R_V"
[7] "Nctx_18L_MS"   "Nctx_18R_MS"   "Nctx_18L_P"
[10] "Nctx_18R_P"    "Nctx_18L_O"    "Nctx_18R_O"
[13] "Nctx_18L_A"    "Nctx_18R_A"    "Nctx_18L_T"
[16] "Nctx_18R_T"    "Nctx_18L_MP"   "Nctx_18R_MP"
[19] "Hipp_18L"      "Hipp_18R"      "Striat_18L"
[22] "Striat_18R"    "Cbllm_18NA"    "Thal_18L"
[25] "Thal_18R"     "Nctx_19L_D"    "Nctx_19R_D"
[28] "Nctx_19L_Or"  "Nctx_19R_Or"  "Nctx_19L_V"
[31] "Nctx_19R_V"   "Nctx_19L_MP"  "Nctx_19R_MP"
[34] "Nctx_19L_MS"  "Nctx_19R_MS"  "Striat_19L"
[37] "Striat_19R"   "Thal_19L"     "Thal_19R"
[40] "Nctx_19L_P"   "Nctx_19R_P"   "Nctx_19L_A"
[43] "Nctx_19R_A"   "Nctx_19L_T"   "Nctx_19R_T"
[46] "Nctx_19L_O"   "Nctx_19R_O"   "Hipp_19L"
[49] "Hipp_19R"     "Amyg_19L_Am"  "Amyg_19R_Am"
[52] "VTA_19L"      "VTA_19R"      "Pons_19L"
[55] "Pons_19R"     "Medulla_19L"  "Medulla_19R"
[58] "Cbllm_19L"    "Cbllm_19R"    "Nctx_21L_D"
[61] "Nctx_21R_D"   "Nctx_21L_V"   "Nctx_21R_V"
[64] "Nctx_21L_MS"  "Nctx_21R_MS"  "Nctx_21L_P"
[67] "Nctx_21R_P"   "Nctx_21L_O"   "Nctx_21R_O"
[70] "Nctx_21L_A"   "Nctx_21R_A"   "Nctx_21L_T"
[73] "Nctx_21R_T"   "Nctx_21L_MP"  "Nctx_21R_MP"
[76] "Hipp_21L"     "Hipp_21R"     "Striat_21L"
[79] "Striat_21R"   "Cbllm_21NA"   "Thal_21L"
[82] "Thal_21R"     "Nctx_23L_D"   "Nctx_23R_D"
[85] "Nctx_23L_V"   "Nctx_23R_V"   "Nctx_23L_MS"
[88] "Nctx_23R_MS"  "Nctx_23L_P"   "Nctx_23R_P"
[91] "Nctx_23L_O"   "Nctx_23R_O"   "Nctx_23L_A"
[94] "Nctx_23R_A"   "Nctx_23L_T"   "Nctx_23R_T"
[97] "Nctx_23L_MP"  "Nctx_23R_MP"  "Hipp_23L"
[100] "Hipp_23R"     "Striat_23L"   "Striat_23R"
[103] "Cbllm_23NA"   "Thal_23L"     "Thal_23R1"
[106] "Pooled_18Control" "Pooled_19Control" "Pooled_19Control"
[109] "Pooled_21Control" "Pooled_23Control"

```

```

#rearrange columns

```

```

dat<-dat1[,c(1:2,17:18,32:33,74:75,97:98, 3:4,26:27,60:61,83:84, 5:6,30:31,62:63,85:86,
7:8,34:35,64:65,87:88, 15:16,44:45, 72:73,95:96, 13:14,42:43, 70:71,93:94, 9:10, 40:41,
66:67, 89:90,11:12,46:47,68:69,91:92, 19:20,48:49,76:77,99:100, 21:22, 36:37,
78:79,101:102, 24:25,38:39,81:82,104:105,23,58:59,80,103)]

```

```

[1] "Transcript.ID"  "gene_assignment" "Nctx_18L_MP"  "Nctx_18R_MP"
[5] "Nctx_19L_MP"    "Nctx_19R_MP"    "Nctx_21L_MP"  "Nctx_21R_MP"
[9] "Nctx_23L_MP"    "Nctx_23R_MP"    "Nctx_18L_D"   "Nctx_18R_D"
[13] "Nctx_19L_D"     "Nctx_19R_D"     "Nctx_21L_D"   "Nctx_21R_D"
[17] "Nctx_23L_D"     "Nctx_23R_D"     "Nctx_18L_V"   "Nctx_18R_V"
[21] "Nctx_19L_V"     "Nctx_19R_V"     "Nctx_21L_V"   "Nctx_21R_V"
[25] "Nctx_23L_V"     "Nctx_23R_V"     "Nctx_18L_MS"  "Nctx_18R_MS"
[29] "Nctx_19L_MS"    "Nctx_19R_MS"    "Nctx_21L_MS"  "Nctx_21R_MS"
[33] "Nctx_23L_MS"    "Nctx_23R_MS"    "Nctx_18L_T"   "Nctx_18R_T"
[37] "Nctx_19L_T"     "Nctx_19R_T"     "Nctx_21L_T"   "Nctx_21R_T"
[41] "Nctx_23L_T"     "Nctx_23R_T"     "Nctx_18L_A"   "Nctx_18R_A"
[45] "Nctx_19L_A"     "Nctx_19R_A"     "Nctx_21L_A"   "Nctx_21R_A"
[49] "Nctx_23L_A"     "Nctx_23R_A"     "Nctx_18L_P"   "Nctx_18R_P"
[53] "Nctx_19L_P"     "Nctx_19R_P"     "Nctx_21L_P"   "Nctx_21R_P"
[57] "Nctx_23L_P"     "Nctx_23R_P"     "Nctx_18L_O"   "Nctx_18R_O"
[61] "Nctx_19L_O"     "Nctx_19R_O"     "Nctx_21L_O"   "Nctx_21R_O"
[65] "Nctx_23L_O"     "Nctx_23R_O"     "Hipp_18L"     "Hipp_18R"

```



```

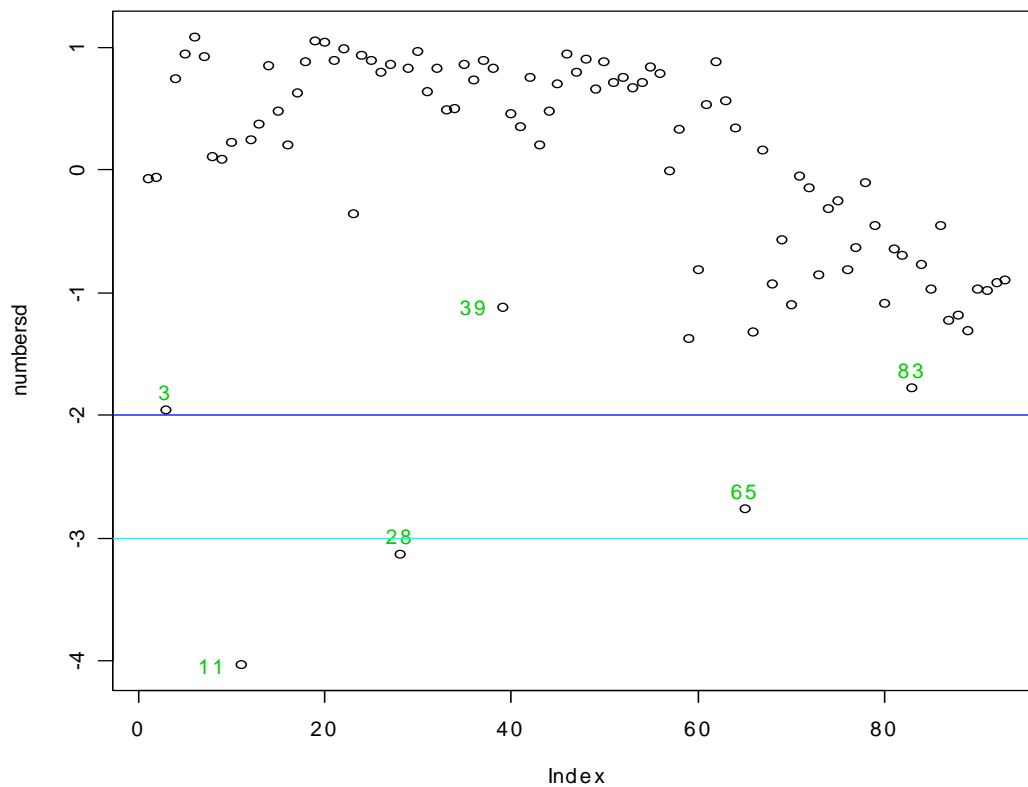
numbersd=(meanIAC-mean(meanIAC))/sdCorr
y=length(IAC[1,])
x=(mean(IAC)*(y^2)-y)/((y^2)-y)
x
#[1] 0.987

```

```

pdf(file="1_nums_d_norm.pdf",width=12,height=12)
plot(numbersd)
abline(h=-2, col=4)
abline(h=-3, col=5)
dev.off()

```



```

SDo<-as.data.frame(cbind(Array=(1:length(numbersd)),Name=names(numbersd),SD=(numbersd)))
SDo[,3]<-as.numeric(as.character(SDo[,3]))
SDo<-SDo[order(SDo[,3]),]
SDoOut<-SDo[SDo[,3]<=(-2),]
SDoOut

```

Array	Name	SD
Nctx_19L_D	11 Nctx_19L_D	-4.033243
Nctx_19R_MS	28 Nctx_19R_MS	-3.134832
Hipp_18L	65 Hipp_18L	-2.758235

```
#outlier exclusion
```

```

to.exclude<-c(11,28,65)

dat3_good<- dat3[,-to.exclude]

write.table(dat3_good, "Yale norm good.csv", sep="," , row.names=F)

datExpr = data.frame(t(dat3_good)) #keep present genes from yale
dim(datExpr)

```

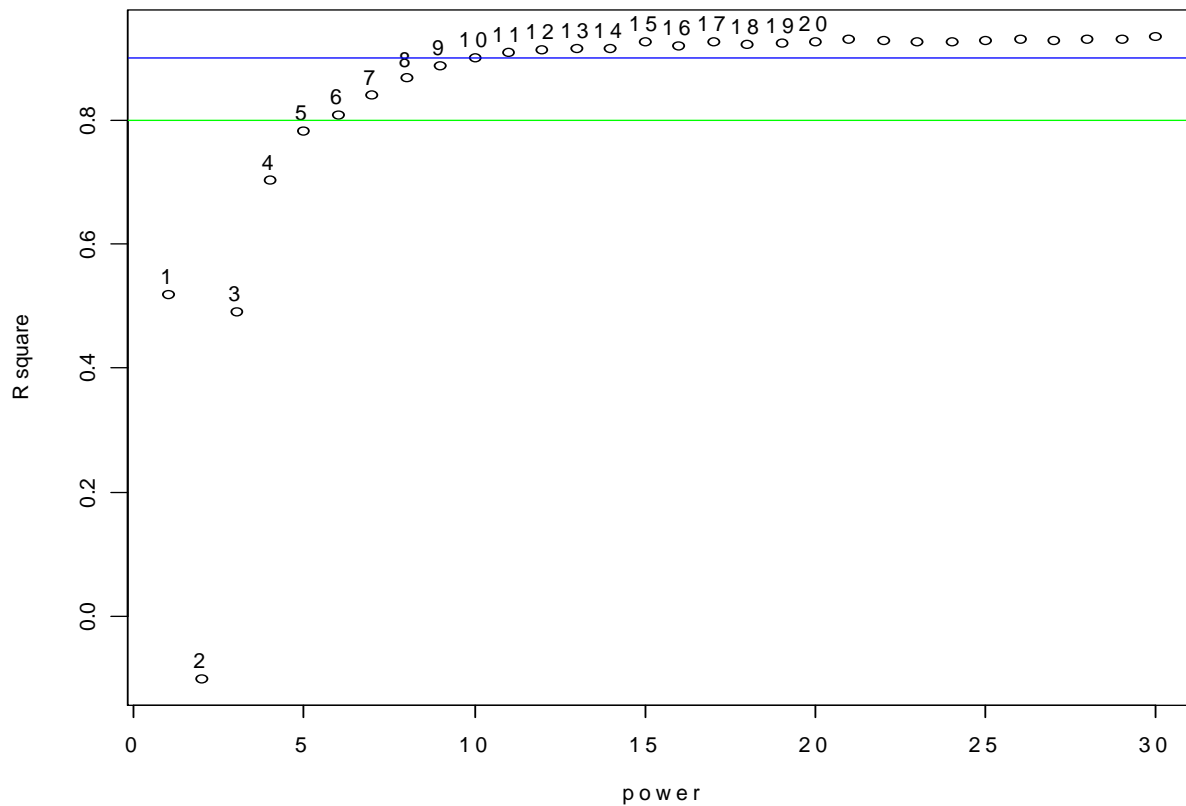
3. choosing the power

```

powers1=c(seq(1,30,by=1))
collect_garbage()
TableSoft=PickSoftThreshold(datExpr, powervector=powers1)[[2]]

```

Power	scale.law.R.2	slope	truncated.R.2	mean.k.	median.k.	max.k.
1	1	0.518	2.210	0.980	4960.000	4.92e+03
2	2	-0.102	-0.156	0.881	2020.000	1.86e+03
3	3	0.491	-0.928	0.921	992.000	8.22e+02
4	4	0.704	-1.270	0.940	543.000	4.02e+02
5	5	0.782	-1.450	0.952	321.000	2.11e+02
6	6	0.809	-1.560	0.951	201.000	1.18e+02
7	7	0.841	-1.620	0.962	131.000	6.84e+01
8	8	0.869	-1.640	0.973	88.900	4.11e+01
9	9	0.889	-1.650	0.982	61.900	2.55e+01
10	10	0.901	-1.660	0.986	44.200	1.63e+01
11	11	0.910	-1.660	0.989	32.200	1.05e+01
12	12	0.913	-1.660	0.990	23.900	6.94e+00
13	13	0.916	-1.660	0.990	18.000	4.66e+00
14	14	0.916	-1.670	0.990	13.800	3.15e+00
15	15	0.926	-1.660	0.993	10.700	2.16e+00
16	16	0.921	-1.670	0.990	8.370	1.50e+00
17	17	0.927	-1.660	0.994	6.620	1.05e+00
18	18	0.923	-1.670	0.992	5.290	7.46e-01
19	19	0.924	-1.660	0.991	4.260	5.32e-01
20	20	0.926	-1.660	0.991	3.460	3.84e-01
21	21	0.930	-1.660	0.994	2.820	2.81e-01
22	22	0.929	-1.650	0.992	2.320	2.06e-01
23	23	0.927	-1.650	0.993	1.920	1.52e-01
24	24	0.926	-1.650	0.993	1.600	1.14e-01
25	25	0.928	-1.650	0.996	1.340	8.53e-02
26	26	0.931	-1.640	0.996	1.130	6.42e-02
27	27	0.929	-1.650	0.994	0.952	4.83e-02
28	28	0.931	-1.640	0.995	0.808	3.70e-02
29	29	0.932	-1.630	0.994	0.688	2.82e-02
30	30	0.936	-1.620	0.996	0.589	2.16e-02



```
plot(TableSoft$Power, TableSoft$scale.law.R.2, xlab="power", ylab="R square")
identify(TableSoft$Power, TableSoft$scale.law.R.2, cex=0.6, col=3 )
abline(h=0.75, col=4)
abline(h=0.8, col=6)
```

4. trying power = 10

```
## power 10
#set here the chosen power
curr.power=10
```

5a. computing the adjacency matrix

```
collect_garbage()
AdjMat = abs(cor(datExpr, use="p"))^curr.power
collect_garbage()
diag(AdjMat)=0
collect_garbage()
dim(AdjMat)
#[1] 17868 17868
```

5b. computing the TOM

```
curr.step<-"5b"
noFirstPass=ncol(datExpr)

collect_garbage()
timestamp()
distTOM <- TOMdist1(AdjMat)
timestamp()

outfile=<-paste(curr.study,"_",curr.step,"_",noFirstPass,"_TOM_", "p",curr.power, sep="")
save(distTOM,file= outfile)
```

5c. clustering TOM

```
curr.step<-"5c"

collect_garbage()
timestamp()
hierTOM <- hclust(as.dist(distTOM),method="average")
timestamp()
collect_garbage()

outfile=<-paste(curr.study,"_",curr.step,"_",noFirstPass,"_HCLUST_", "p",curr.power,
sep="")
save(hierTOM,file= outfile)
```

6. cutting the tree

```
curr.step<-"6"

#trying eight different cutoffs: 0.995, 0.990, 0.985 and 0.980, 0.975, 0.97, 0.96, 0.965

mydeepSplit      = TRUE
myminModuleSize  = 9
```

```
outfile=<-paste(curr.study,"_",curr.step, "_", noFirstPass,
"_InitialModulesTesting_", "p",curr.power, ".pdf",sep="")
```

#this loop outputs the modules and grey genes for 4 different cutoffs. More can be added

```
pdf(file=outfile)
for (cutoffs in c(0.995,0.990,0.985,0.980, 0.975, 0.97, 0.965, 0.96)) {
  myheightcutoff = cutoffs

  colorhl=cutreeDynamic(hierclust= hierTOM, deepSplit=mydeepSplit,
maxTreeHeight=myheightcutoff, minModuleSize=myminModuleSize)

  par(mfrow=c(2,1),mar=c(2,2,2,2))
  plot(hierTOM,
main=paste(outfile,"_cutoff: ",cutoffs,
  "_Modules=",length(unique(colorhl)),
  "_NotGrey=", length(colorhl[!colorhl=="grey"])
,sep=""),labels=F,cex.main=0.8)
  abline(h=myheightcutoff,col="red",lwd=1)
  hclustplot1(hierTOM,colorhl,title1=paste("DTC modules",myheightcutoff,
"min",myminModuleSize,"mds=",mydeepSplit))

  cat("cutoff=",cutoffs," Modules=",length(unique(colorhl)),
NotGreyGenes",length(colorhl[!colorhl=="grey"]),"\n")
} #end cutoff testing

dev.off()
```

```
cutoff= 0.995 Modules= 188 NotGreyGenes 14790
```

```

cutoff= 0.99 Modules= 193 NotGreyGenes 12946
cutoff= 0.985 Modules= 186 NotGreyGenes 11059
cutoff= 0.98 Modules= 163 NotGreyGenes 9461
cutoff= 0.975 Modules= 156 NotGreyGenes 8281
cutoff= 0.97 Modules= 133 NotGreyGenes 6922
cutoff= 0.965 Modules= 98 NotGreyGenes 5890
cutoff= 0.96 Modules= 90 NotGreyGenes 5111

```

7a. choosing a cut threshold

```
curr.step="7a"
```

```
curr.cutoff<-0.98
```

```

mydeepSplit      = TRUE
myminModuleSize  = 9
myheightcutoff   = curr.cutoff

```

```

#cutreeDynamic function
colorh1=cutreeDynamic(hierclust= hierTOM, deepSplit=mydeepSplit,
maxTreeHeight=myheightcutoff, minModuleSize=myminModuleSize)

```

```
data.frame(table(colorh1))
```

```

      colorh1 Freq
plum1      1220
darkseagreen2 899
springgreen1  553

lemonchiffon      10
azure              10
grey             8407

```

```
# building the table for the color names
```

```

PSID=dat$Transcript.ID
datout1=data.frame(PSID,colorh1)
colnames(datout1)=c("PSID","colorh1")

```

```

outfilename<-paste(curr.study,"_",curr.step,
"_InitialModules_", "cut",curr.cutoff,"_p",curr.power,".csv",sep="")

```

```
write.table(datout1, paste(outfilename),sep=",")
```

7b. computing and clustering the PCs

```
curr.step="7b"
```

```

pcs = ModulePrinComps1(datexpr=as.matrix(datExpr), couleur=as.character(colorh1))
distCorPCs = 1-abs(cor(pcs[[1]],use="p"))
distCorPCs = ifelse(is.na(distCorPCs), 0, distCorPCs)
h1PCs      = hclust(as.dist(distCorPCs),method="a")

```

```
##PDF output
```

```

outfilename<-
paste(curr.study,"_",curr.step,"_",noFirstPass,"_InitialModules_", "cut",curr.cutoff,"_p",c
urr.power,"_PCs.pdf",sep="")

```

```

pdf(file= paste(outfilename,sep=""),bg="transparent",width=21,height=12)
par(mfrow=c(3,1),mar=c(2,2,2,2))

```

```

#first section: tree
plot(hierTOM,main=outfilename,labels=F)

```

```

#second section: color codes for modules
hclustplot1(hierTOM,colorh1,title=paste("DTC modules,",myheightcutoff,
"min",myminModuleSize,"mds=",mydeepSplit))

```

```

#third section: clustering of PCs"
plot(h1PCs, xlab="",ylab="",main="",sub="")

```

```
abline(h=0.1, col=3)
abline(h=0.2, col=4)
dev.off()
```

7c. permutations

```
#this function works with a matrix with dim = (firstPass x firstPass)
#1) for each module as identified by the threshold (in this case 0.990), takes the number
    of genes in that module,
#2) computes the average TO,
#3) selects the same number of random genes in the network (within the firstPass Genes),
#4) computes the average TO in that group
#5) does this 5,000 times and comes up with an empirical pvalue that tells us how likely
    is to have a group of the same size and with the same average TO in the network
```

```
curr.step="7c"
timestamp()
permutations=5000
InitialModuleNumbers=ModuleQCnoGrey(datExpr1=datExpr, couleur=colorh1, distTOM1=distTOM, hier
TOM1=hierTOM, pcs1=pcs, perms=permutations)
timestamp()

outfilename<-paste(curr.study, "_", curr.step,
"_InitialModules_", "cut", curr.cutoff, "_p", curr.power, "_permutations", permutations, ".csv", s
ep="")

write.table(InitialModuleNumbers,
paste(outfilename, sep=""), sep=" ", row.names=F, col.names=T)

collect_garbage()

save.image("Perm")
#load("Perm")
```

7d. setting to grey the non-significant modules

```
#based on the permutations, we decide to keep just the modules that are significant
according to the empirical p-value.
## Going to use a cutoff of p < 0.01; i.e., if the p.value for a module having an average
TO is < .01, we'll keep it; set the rest to grey.

keptmodules=c(as.character(InitialModuleNumbers$Module[as.numeric(InitialModuleNumbers$P.v
alue)<.01]))
length(keptmodules)
#[1] 29

#creates the new vector with non-significant modules assigned to grey:
#1) creates an all=grey vector
colorh2=rep("grey", length(colorh1))

data.frame(table(colorh1))

keptmodvec=is.element(colorh1, keptmodules)

#assigns the module colors to the genes in kept modules
colorh2[keptmodvec]=as.character(colorh1[keptmodvec])
colorh1=colorh2

#the total will be # modules +1 (grey)
data.frame(table(colorh2))
```

#8. Re-building the network without the grey genes

```
curr.step="8"

greyvec=colorh1=="grey"

datExpr2=datExpr[,!greyvec]
dim(datExpr2)
#[1] 90 9641

# the genes in the second network (noSecondPass) are 2558

noFirstPass=ncol(datExpr)
noSecondPass=ncol(datExpr2)
```

#8a. computing the adjacency matrix

```
AdjMat2 = abs(cor(datExpr2,use="p"))^curr.power
diag(AdjMat2)=0
collect_garbage()
dim(AdjMat2)
# [1] 9641 9641

PSID2=dat1$Transcript.ID[!greyvec]
#Gene2=dat1$Symbol[dat1$p995>0][!greyvec]
```

8b. computing the TOM

```
curr.step="8b"

timestamp()
distTOM2 <- TOMdist1(AdjMat2)
timestamp()

save(distTOM2,file="DRG_34_6218_TOM_p8_99cut_min9_p01greyRemove_2558")

collect_garbage()
```

8c. clustering TOM

```
curr.step="8c"
hierTOM2 <- hclust(as.dist(distTOM2),method="average")
save(hierTOM2,file="DRG_34_6218_HierClust_p8_99cut_min9_p01greyRemove_2558")

collect_garbage()
```

9. cutting the tree

```
curr.step<-"9"

#trying four different cutoffs: 0.990, 0.985, 0.980, 0.975, 0.97, 0.965, 0.96

mydeepSplit      = TRUE
myminModuleSize  = 9

outfilename<-
paste(curr.study,"_",curr.step,"_",noFirstPass,"_ModuleSecondTesting_", "p",curr.power,"greyRemove",noSecondPass,".pdf",sep="")
```

#this loop outputs the modules and grey genes for 4 different cutoffs. More can be added

```

pdf(file= paste(outfilename,sep=""))
for (cutoffs in c(0.995, 0.990,0.985,0.980, 0.975, 0.970, 0.965, 0.960, 0.955, 0.950,
0.945, 0.940, 0.935, 0.930, 0.925, 0.920, 0.915, 0.910, 0.905, 0.900)) {
  myheightcutoff = cutoffs

  colorh1=cutreeDynamic(hierclust= hierTOM2, deepSplit=mydeepSplit,
maxTreeHeight=myheightcutoff, minModuleSize=myminModuleSize)
par(mfrow=c(2,1),mar=c(2,2,2,2))
  plot(hierTOM2,
main=paste(outfilename,"_cutoff: ",cutoffs,
  "_Modules=",length(unique(colorh1)),
  "_NotGrey=", length(colorh1[!colorh1=="grey"])]
,sep=""),labels=F,cex.main=0.8)
abline(h=myheightcutoff,col=2,lwd=0.8)
  hclustplot1(hierTOM2,colorh1,title1=paste("DTC
modules",myheightcutoff, "min",myminModuleSize,"mds=",mydeepSplit))
  cat("cutoff=",cutoffs," Modules=",length(unique(colorh1)),
  NotGreyGenes",length(colorh1[!colorh1=="grey"]),"\\n")
} #end cutoff testing
dev.off()

```

9a. choosing a cut threshold

```
curr.step="9a"
```

```

myheightcutoff =0.98
mydeepSplit = TRUE
myminModuleSize = 9

```

```

colorh1=cutreeDynamic(hierclust=hierTOM2, deepSplit=mydeepSplit,
maxTreeHeight=myheightcutoff, minModuleSize=myminModuleSize)

```

```

length(unique(colorh1))
#[1] 146

```

```

outfilename<-
paste(curr.study,"_",curr.step,"_",noFirstPass,"_", "p",curr.power,"greyRemove",noSecondPas
s,"_ChosenCut=",myheightcutoff,".pdf",sep="")

```

```
pdf(file= paste(outfilename,sep=""),bg="transparent",width=21,height=12)
```

```

par(mfrow=c(2,1),mar=c(2,2,2,2))
plot(hierTOM2,main=outfilename,labels=F,cex.main=0.8)
hclustplot1(hierTOM2,colorh1,title1=paste("DTC modules",myheightcutoff,
"min",myminModuleSize,"mds=",mydeepSplit))
dev.off()

```

9b. computing and clustering the PCs

```
curr.step="9b "
```

```

pcs = ModulePrinComps1(datexpr=as.matrix(datExpr2), couleur=as.character(colorh1))
distCorPCs = 1-abs(cor(pcs[[1]],use="p"))
distCorPCs = ifelse(is.na(distCorPCs), 0, distCorPCs)
hlPCs = hclust(as.dist(distCorPCs),method="a")

```

```
#This function gives the list of module colors _and_ their position in the tree.
```

```
Rd2ModuleNumbers=ModuleQCnoPerm(datExpr1=datExpr2,couleur=colorh1,distTOM1=distTOM2,hierTO
M1=hierTOM2,pcs1=pcs)
```

```
#table output
```

```
write.table(Rd2ModuleNumbers, file="ModuleNumbers_SecondPass_985cut_6218.csv", sep=",", row.names=F, col.names=T)
collect_garbage()
```

```
mod.pos<- Rd2ModuleNumbers[,c(1,2,8)]
mod.pos[,1]<- (as.character(mod.pos[,1]))
mod.pos<-mod.pos[order(mod.pos[,1]),]
```

```
h1PCs$labels
```

```
mod.pos
```

```
#just to check
```

```
new.lab<-paste(mod.pos[,3], mod.pos[,1], mod.pos[,2], sep="_")
new.lab->h1PCs$labels
```

#PDF output

```
outfile<-
paste(curr.study, "_", curr.step, "_", noFirstPass, "_", "p", curr.power, "greyRemove", noSecondPass,
"_ChosenCut=", myheightcutoff, "PCs.pdf", sep="")
```

```
pdf(file= paste(outfile, sep=""), bg="transparent", width=21, height=12)
```

```
par(mfrow=c(3,1), mar=c(2,2,2,2))
```

```
#first section: tree
```

```
plot(hierTOM2, main=outfile, labels=F)
```

```
#second section: color codes for modules
```

```
hclustplot1(hierTOM2, colorh1, title1=paste("DTC modules", myheightcutoff, "min", myminModuleSize))
```

```
#third section: clustering of PCs
```

```
plot(h1PCs, xlab="", ylab="", main="", sub="")
```

```
abline(h=0.1, col=3)
```

```
abline(h=0.2, col=5)
```

```
dev.off()
```

```
#gene IDs and module membership (length=noSecondPass)
```

```
datout2=data.frame(PSID[!greyvec], colorh1)
```

```
colnames(datout2)=c("PSID", "colorh1")
```

```
write.table(datout2, "ModuleNumbers_SecondPass_singleGenes_premerge_cut98.csv", sep=",", row.names=F)
```

```
save.image("before_merge")
```

```
# for merging, refer to "color1_merge.doc"
```

```
#after first round merge
```

10. Merging the modules

```
#2
```

```
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="orchid4")
```

```
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="navajowhite4")
```

```
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="lightgoldenrod")
```

```
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="chocolate2")
```

```
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="navy")
```

```
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="orange3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="cyan2")

#11
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow1", minorclusterColor="palegoldenrod")
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow1", minorclusterColor="thistle2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow1", minorclusterColor="white")
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow1", minorclusterColor="saddlebrown")

#16-19
colorh1 = merge2Clusters(colorh1, mainclusterColor="darkorange2", minorclusterColor="aquamarine3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="darkorange2",
minorclusterColor="lavenderblush3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="darkorange2", minorclusterColor="magenta4")

#21-24
colorh1 = merge2Clusters(colorh1, mainclusterColor="tomato3", minorclusterColor="bisque3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="tomato3", minorclusterColor="indianred4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="tomato3", minorclusterColor="navajowhite2")

#26-28
colorh1 = merge2Clusters(colorh1, mainclusterColor="orangered", minorclusterColor="lemonchiffon1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="orangered",
minorclusterColor="lightgoldenrodyellow")

#30-31
colorh1 = merge2Clusters(colorh1, mainclusterColor="coral2", minorclusterColor="darkkhaki")

#32-36
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow2", minorclusterColor="salmon4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow2", minorclusterColor="plum4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow2", minorclusterColor="tan1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow2", minorclusterColor="yellow3")

#37-40
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="springgreen3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="hotpink2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="darkslategrey")

#41-42
colorh1 = merge2Clusters(colorh1, mainclusterColor="lightblue4", minorclusterColor="orangered3")

#43-45
colorh1 = merge2Clusters(colorh1, mainclusterColor="red1", minorclusterColor="mediummorchid2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="red1", minorclusterColor="darkorchid4")

#46-48
colorh1 = merge2Clusters(colorh1, mainclusterColor="hotpink", minorclusterColor="paleturquoise")
colorh1 = merge2Clusters(colorh1, mainclusterColor="hotpink", minorclusterColor="skyblue1")

#49-52
colorh1 = merge2Clusters(colorh1, mainclusterColor="orange2", minorclusterColor="indianred2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="orange2", minorclusterColor="darkgoldenrod")
colorh1 = merge2Clusters(colorh1, mainclusterColor="orange2", minorclusterColor="goldenrod1")

#53-56
colorh1 = merge2Clusters(colorh1, mainclusterColor="lightyellow", minorclusterColor="dodgerblue2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="lightyellow", minorclusterColor="sandybrown")
colorh1 = merge2Clusters(colorh1, mainclusterColor="lightyellow",
minorclusterColor="darkolivegreen3")

#57-61
colorh1 = merge2Clusters(colorh1, mainclusterColor="deepskyblue1",
minorclusterColor="mediumorchid")
colorh1 = merge2Clusters(colorh1, mainclusterColor="deepskyblue1",
minorclusterColor="mediumturquoise")
colorh1 = merge2Clusters(colorh1, mainclusterColor="deepskyblue1", minorclusterColor="turquoise1")
```

```
colorh1 = merge2Clusters(colorh1, mainclusterColor="deepskyblue1", minorclusterColor="slategray")

#62-63
colorh1 = merge2Clusters(colorh1, mainclusterColor="royalblue4", minorclusterColor="cornsilk1")

#64-72
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="antiquewhite")
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="mediumpurple2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="lightsteelblue3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="olivedrab4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="honeydew3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="burlywood3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="cadetblue3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew", minorclusterColor="turquoise3")

#73-76
colorh1 = merge2Clusters(colorh1, mainclusterColor="brown1", minorclusterColor="lightgrey")
colorh1 = merge2Clusters(colorh1, mainclusterColor="brown1", minorclusterColor="dimgray")
colorh1 = merge2Clusters(colorh1, mainclusterColor="brown1", minorclusterColor="palegreen2")

#77-80
colorh1 = merge2Clusters(colorh1, mainclusterColor="snow", minorclusterColor="lightsalmon")
colorh1 = merge2Clusters(colorh1, mainclusterColor="snow", minorclusterColor="turquoise4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="snow", minorclusterColor="orchid2")

#81-85
colorh1 = merge2Clusters(colorh1, mainclusterColor="red2", minorclusterColor="beige")
colorh1 = merge2Clusters(colorh1, mainclusterColor="red2", minorclusterColor="hotpink3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="red2", minorclusterColor="lightskyblue")
colorh1 = merge2Clusters(colorh1, mainclusterColor="red2", minorclusterColor="mediumorchid4")

#86-86-89
colorh1 = merge2Clusters(colorh1, mainclusterColor="lightpink3", minorclusterColor="cyan1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="lightpink3",
minorclusterColor="darkslategray1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="lightpink3",
minorclusterColor="mediumspringgreen")

#90-91
colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew2", minorclusterColor="lemonchiffon")

#92-96
colorh1 = merge2Clusters(colorh1, mainclusterColor="green4", minorclusterColor="plum1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="green4", minorclusterColor="royalblue1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="green4", minorclusterColor="firebrick1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="green4", minorclusterColor="peachpuff1")

#97-101
colorh1 = merge2Clusters(colorh1, mainclusterColor="orangered4", minorclusterColor="mintcream")
colorh1 = merge2Clusters(colorh1, mainclusterColor="orangered4", minorclusterColor="lightsalmon4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="orangered4", minorclusterColor="burlywood4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="orangered4", minorclusterColor="thistle")

#102-106
colorh1 = merge2Clusters(colorh1, mainclusterColor="green1", minorclusterColor="deeppink3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="green1", minorclusterColor="violet")
colorh1 = merge2Clusters(colorh1, mainclusterColor="green1", minorclusterColor="tan3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="green1", minorclusterColor="orange")

#107-108
colorh1 = merge2Clusters(colorh1, mainclusterColor="seashell2", minorclusterColor="darkgoldenrod3")

#109-112
colorh1 = merge2Clusters(colorh1, mainclusterColor="gold3", minorclusterColor="antiquewhite3")
```

```

colorh1 = merge2Clusters(colorh1, mainclusterColor="gold3", minorclusterColor="green")
colorh1 = merge2Clusters(colorh1, mainclusterColor="gold3", minorclusterColor="lightslategray")

#113-121
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="orange4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="lavenderblush4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="cadetblue4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="coral")
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="chocolate1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="navajowhite3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="lavenderblush1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="hotpink4")

#122-129
colorh1 = merge2Clusters(colorh1, mainclusterColor="rosybrown1", minorclusterColor="whitesmoke")
colorh1 = merge2Clusters(colorh1, mainclusterColor="rosybrown1", minorclusterColor="magenta2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="rosybrown1", minorclusterColor="darkslategray")
colorh1 = merge2Clusters(colorh1, mainclusterColor="rosybrown1", minorclusterColor="deepskyblue4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="rosybrown1", minorclusterColor="snow4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="rosybrown1", minorclusterColor="slategray2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="rosybrown1", minorclusterColor="tan2")

#130-131
colorh1 = merge2Clusters(colorh1, mainclusterColor="salmon3", minorclusterColor="lemonchiffon4")

#132-133
colorh1 = merge2Clusters(colorh1, mainclusterColor="palegreen1", minorclusterColor="mediumpurple3")

#134-137
colorh1 = merge2Clusters(colorh1, mainclusterColor="mediumpurple", minorclusterColor="turquoise")
colorh1 = merge2Clusters(colorh1, mainclusterColor="mediumpurple", minorclusterColor="goldenrod4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="mediumpurple", minorclusterColor="blue1")

#138-141
colorh1 = merge2Clusters(colorh1, mainclusterColor="wheat2", minorclusterColor="honeydew4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="wheat2", minorclusterColor="mediumseagreen")
colorh1 = merge2Clusters(colorh1, mainclusterColor="wheat2", minorclusterColor="palevioletred2")

#142-146
colorh1 = merge2Clusters(colorh1, mainclusterColor="steelblue1",
minorclusterColor="lightgoldenrod1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="steelblue1", minorclusterColor="dodgerblue3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="steelblue1", minorclusterColor="lightyellow3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="steelblue1", minorclusterColor="royalblue")

#2nd
colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow4", minorclusterColor="azure")
colorh1 = merge2Clusters(colorh1, mainclusterColor="tomato3", minorclusterColor="mediumaquamarine")
colorh1 = merge2Clusters(colorh1, mainclusterColor="tomato3", minorclusterColor="darkseagreen1")

colorh1 = merge2Clusters(colorh1, mainclusterColor="coral2", minorclusterColor="darkseagreen4")

colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="lightblue4")
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="red1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="mediumorchid2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="hotpink")
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="orange2")
colorh1 = merge2Clusters(colorh1, mainclusterColor="blue", minorclusterColor="lightyellow")

colorh1 = merge2Clusters(colorh1, mainclusterColor="snow", minorclusterColor="brown1")

```

```

colorh1 = merge2Clusters(colorh1, mainclusterColor="honeydew2", minorclusterColor="green4")

colorh1 = merge2Clusters(colorh1, mainclusterColor="gold3", minorclusterColor="seagreen")

colorh1 = merge2Clusters(colorh1, mainclusterColor="wheat2", minorclusterColor="palegreen1")
colorh1 = merge2Clusters(colorh1, mainclusterColor="wheat2", minorclusterColor="salmon3")
colorh1 = merge2Clusters(colorh1, mainclusterColor="wheat2", minorclusterColor="mediumpurple")
colorh1 = merge2Clusters(colorh1, mainclusterColor="wheat2", minorclusterColor="steelblue1")

#3 change colors
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="coral")

colorh1 = merge2Clusters(colorh1, mainclusterColor="violet", minorclusterColor="royalblue4")

colorh1 = merge2Clusters(colorh1, mainclusterColor="yellow", minorclusterColor="yellow1")

colorh1 = merge2Clusters(colorh1, mainclusterColor="purple", minorclusterColor="yellow2")

colorh1 = merge2Clusters(colorh1, mainclusterColor="seashell", minorclusterColor="seashell2")

colorh1 = merge2Clusters(colorh1, mainclusterColor="black", minorclusterColor="tomato3")

colorh1 = merge2Clusters(colorh1, mainclusterColor="pink", minorclusterColor="lightpink3")

colorh1 = merge2Clusters(colorh1, mainclusterColor="red", minorclusterColor="red2")

colorh1 = merge2Clusters(colorh1, mainclusterColor="green", minorclusterColor="green1")

colorh1 = merge2Clusters(colorh1, mainclusterColor="orange", minorclusterColor="orangered4")

colorh1 = merge2Clusters(colorh1, mainclusterColor="brown", minorclusterColor="rosybrown1")

```

```

#4 change colors
colorh1 = merge2Clusters(colorh1, mainclusterColor="seagreen", minorclusterColor="coral2")

colorh1 = merge2Clusters(colorh1, mainclusterColor="gold3", minorclusterColor="seashell")

colorh1 = merge2Clusters(colorh1, mainclusterColor="pink", minorclusterColor="honeydew2")

colorh1 = merge2Clusters(colorh1, mainclusterColor="deepskyblue1", minorclusterColor="violet")

```

```

#5 change colors
colorh1 = merge2Clusters(colorh1, mainclusterColor="purple", minorclusterColor="seagreen")

colorh1 = merge2Clusters(colorh1, mainclusterColor="salmon", minorclusterColor="honeydew")

```

10c. computing and clustering the PCs

```
curr.step<-"10c5"
```

```

pcs = ModulePrinComps1(datexpr=as.matrix(datExpr2), couleur=as.character(colorh1))
distCorPCs = 1-abs(cor(pcs[[1]],use="p"))
distCorPCs = ifelse(is.na(distCorPCs), 0, distCorPCs)
h1PCs      = hclust(as.dist(distCorPCs),method="a")

```

```

#This function gives the list of module colors _and_ their position in the tree.
Rd6ModuleNumbers=ModuleQCnoPerm(datExpr1=datExpr2,couleur=colorh1,distTOM1=distTOM2,hierTOM1=hierTOM2,pcs1=pcs)

```

```

mod.pos<- Rd6ModuleNumbers[,c(1,2,8)]
mod.pos[,1]<- (as.character(mod.pos[,1]))
mod.pos<-mod.pos[order(mod.pos[,1]),]

```

```

h1PCs$labels
mod.pos
#just to check

new.lab<-paste(mod.pos[,3],mod.pos[,1], mod.pos[,2],sep="_")
new.lab->h1PCs$labels

#write.table(pcs[[1]], "drg PC1 for module trend.csv", sep=",")

outfile<-
paste(curr.study,"_",curr.step,"_",noFirstPass,"_", "p",curr.power,"greyRemove",noSecondPas
s,"_ChosenCut=",myheightcutoff,"PCs_afterMerging.pdf",sep="")

pdf(file= paste(outfile,sep=""),bg="transparent",width=21,height=12)

par(mfrow=c(3,1),mar=c(2,2,2,2))

plot(hierTOM2,main=outfile,labels=F)

hclustplot1(hierTOM2,colorh1,title1=paste("DTC modules,",myheightcutoff,
"min",myminModuleSize))

plot(h1PCs, xlab="",ylab="",main="",sub="")
abline(h=0.1,col=3)
abline(h=0.2,col=6)

dev.off()

save.image("round5")

```

11. Data output

```

#read in annotation file
ann<-read.csv("for output.csv", header=T)

dat2<-cbind(dat1$Transcript.ID, dat)
colnames(dat2)[1]<-"Transcript.ID"

dat.new<-merge(ann, dat2, by.x="TranscriptID", by.y="Transcript.ID")

PSID2=dat.new$TranscriptID[!greyvec]
Accession=dat.new$Accession[!greyvec]

Gene2=dat.new$Symbol[!greyvec]

datout33=data.frame(PSID[!greyvec],colorh1)
colnames(datout33)=c("PSID","colorh1")
write.table(datout33,"modules_postmerge_test.csv",sep="," ,row.names=F)

collect_garbage()
alldegrees=DegreeInOutMO(datExpr1=datExpr2,adj1=AdjMat2,couleur=colorh1,pcs1=pcs)
collect_garbage()
dim(alldegrees)
meanExpr=apply(datExpr2,2,mean)
datout=data.frame(PSID2, Gene2, Accession, colorh1, alldegrees, meanExpr)

write.table(datout,file="Yale_Modules_k&e_info.csv",sep="," ,row.names=F,col.names=T)

FinalModuleNumbers=ModuleQCnoPerm(datExpr1=datExpr2,couleur=colorh1,distTOM1=distTOM2,hier
TOM1=hierTOM2,pcs1=pcs)

```

```

write.table(FinalModuleNumbers,file="FinalModuleNumbers_9461.csv",sep=" ",row.names=F,col.
names=T)
collect_garbage()

#read out PCS
rownames(pcs[[1]])<-rownames(datExpr)
write.table(pcs[[1]], "Yale PC1 for module trend.csv", sep=" ")

# make file with expression values from dat1(original data)
dat11<-subset(dat.new, dat.new$TranscriptID %in% PSID2)
write.table(dat11, "forVisant.csv", sep=" ", row.names=F, col.names=T)

color<-datout[,4]
genes<-datout[,2]
datvis<-dat11[,c(4:93)]
rownames(datvis)<- datout$PSID2
datEt<-t(datvis)
#output for visAnt
## This file returns the overall strongest connections within a module for one group of
subjects

visantPrepOverall <- function(colorFinal2, moduleColor, datExprrest, genes, numInt)
{

## USER inputs
# colorFinal2 = vector with module association for each probe
# moduleColor = color of the module... should be a member of colorFinal2
# datExprrest = expression data for the genes corresponding to cIndex
# genes = list of genes that correspond to the probes
# numInt = number of interactions to output to the visant file

cIndex = (colorFinal2==moduleColor)
datExpr=datExprrest[,cIndex]
AdjMat =abs(cor(datExpr, use="p"))^curr.power
diag(AdjMat)=0
Degree <- apply(AdjMat,1,sum)
Degree = Degree/max(Degree)
meanExpr=apply(datExpr,2,mean)
ProbeSet=colnames(datExpr)
GeneSet=genes[cIndex];
Module=rep(moduleColor,length(meanExpr))

## This file summarizes intramodular connectivity and expression for each gene in each
group:

write.table( cbind(ProbeSet,GeneSet,meanExpr,Degree,Module), file=
paste(moduleColor,"_connectivityOverall.csv",sep=" ")
,sep=" ",row.names=F, col.names= c("probes","genes","meanExpr","kin","module"))

## TOM matrix

distTOM <- TOMdist1(AdjMat)
simTOM = 1-distTOM
diag(simTOM)=0
simTOMcutoff = simTOM

## Correlation matrix

Pearson = cor(datExpr ,use="p")
diag(Pearson)=0

```

```

## Dynamically determine the appropriate cutoff
cutoff = 0.24
len     = 10000
dir     = "increase"
loops  = 0
split   = 0.01
numInt  = numInt+100
while(len>100){
  loops = loops+1
  if (dir == "increase") { cutoff = cutoff+split; }
  if (dir == "decrease") { cutoff = cutoff-split; }
  indices = (simTOMcutoff>cutoff)
  len = sum(sum(indices))
  if (len < numInt) {dir = "decrease";}
  if (len >=numInt) {dir = "increase";}
  len = abs(len-numInt)
  if (loops>500){ len=0;}
  if ((loops%%100)==0){ split = split/2; }
}
write(c(loops,cutoff,len),"")

## Output using cutoffs:

for(i in c(1) ) {
indices = (simTOMcutoff[i,]>cutoff)
datout=data.frame(
rep(GeneSet[i], length(ProbeSet[indices])),
GeneSet[indices],rep(0, length(ProbeSet[indices])),
rep("color", length(ProbeSet[indices])),
simTOM[i,][indices], Pearson[i,][indices])
}

for(i in seq(2,length(ProbeSet),by=1)) {
#for (i in 2){
indices = (simTOMcutoff[i,]>cutoff)
datout=data.frame(rbind(datout,data.frame(
rep(GeneSet[i], length(ProbeSet[indices])),
GeneSet[indices],rep(0, length(ProbeSet[indices])),
rep("color", length(ProbeSet[indices])),
simTOM[i,][indices], Pearson[i,][indices])))
}

write.table(datout,file=paste(moduleColor,"_visantOverall.csv",sep=""),
sep="," ,row.names=F, col.names=c("gene1", "gene2", "zero", "color", "TO", "Correlation"))

}

# to call function
#write("START","")
#visantPrepOverall(color, "orange2", datEt, genes, 200)
#write("DONE","")
# color is colorh1 in k&e file
# datEt(transposed) is expression data with probeID in k&e, with rownames PSID,
GeneSymbol, #Colorh1, and expression values for arrays
# datEt can be made from k&e and original expression (dat1)
# "pink1" is one of the module colors
# gene is gene symbols, it is datEt$GeneSymbol
# 1000 is the inteactions desired, can be any values

```

```

write("START","")
visantPrepOverall(color, "yellow4", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "yellow", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "darkorange2", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "black", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "orangered", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "purple", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "blue", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "deepskyblue1", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "salmon", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "snow", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "red", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "pink", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "orange", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "green", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "gold3", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "brown", datEt, genes, 3000)
write("DONE","")
visantPrepOverall(color, "wheat2", datEt, genes, 3000)
write("DONE","")
write("END","")

```

```

#visantPrepOverall(color, "seagreen", datEt, genes, 3000)
#write("DONE","")

```

```
#save.image("big")
```

12. Heatmaps

```
#since the samples are many and the names cannot be read, need sort the samples and
colorcode them
```

```
#length(which((regex("AD",names))>0))
```

```

rownames(datExpr)->sampleNames
mod.colors<- sort(as.character(unique(colorh1)))
bar_col<-
c(rep("1",44),rep("2",2),rep("3",36),rep("4",13),rep("5",41),rep("6",3),rep("7",1))

```

```
pdf(file="modules_power9.pdf",width=21,height=12)
```

```

for (mod.hea in 1:length(mod.colors)) {

whichmodule<-mod.colors[mod.hea]

par(mfrow=c(2,1))

par(mfrow=c(2,1))
par(mar=c(2,3,2.5,3))
par(oma=c(4,0,2,0))
datcombined=datExpr2[,colorh1==whichmodule]
datcombined=datExpr2[order(sampleNames),colorh1==whichmodule]

plot.mat(t(scale(datcombined)),main= whichmodule)->x

#names=dimnames(datExpr2)[[1]]

#names=sampleNames[order(sampleNames)]
datstv=pcs$PrinComps[mod.hea]
#datstv_sort<-datstv[order(sampleNames),]
#whichmodule="mediumorchid2"

#xpos= seq=(6.2,162.0,1)
#width=0.85,space=0.185

barplot(datsv,col=bar_col,xlim=c(6.2,162.0))->x

axis(side=1,at=x,labels=F,tick=F)
box()
mtext(names,1,at=x,cex=1,adj=1.2,las=3)

}
dev.off()

#procedure to get the x.lim values:
1) assign the barplot to x
2) compute mean (x)
3) set the two values to have the same mean as x, but slightly smaller than the range
4) adjust the range by equal (but opposite) intervals from both sides: increasing the
left-side number will move the plot to the left
5) x can be used to set the text

save.image("before heat") #small module

pdf(file="wheat2.pdf",width=21,height=12)
par(mfrow=c(2,1))
whichmodule="wheat2"
par(mfrow=c(2,1))
par(mar=c(2,3,2.5,3))
par(oma=c(4,0,2,0))
datcombined=datExpr2[,colorh1==whichmodule]
plot.mat(t(scale(datcombined)),main="wheat3(636)")
names=dimnames(datExpr2)[[1]]
datstv=pcs$PrinComps$PCwheat2
whichmodule="wheat2"
xpos=seq(0.5,89.5,1.0)
barplot(datsv,col=whichmodule,xlim=c(3.5,86.5),width=0.85,space=0.175)
axis(side=1,at=xpos,labels=F,tick=T)
box()
mtext(names,1,at=xpos,cex=0.9,adj=1.2,las=3)
dev.off()

```

```

# CMD
timestamp()
pdf(file="CMD.pdf",width=21,height=12)
cmd1=cmdscale(as.dist(distTOM2),4)
par(mfrow=c(2,3))
plot(cmd1[,c(1,2)], col= as.character(colorh1) )
plot(cmd1[,c(1,3)], col= as.character(colorh1) )
plot(cmd1[,c(1,4)], col= as.character(colorh1) )
plot(cmd1[,c(2,3)], col= as.character(colorh1) )
plot(cmd1[,c(2,4)], col= as.character(colorh1) )
plot(cmd1[,c(3,4)], col= as.character(colorh1) )
dev.off()
timestamp()

# MDS plot
# MDS plot from different angles
pdf(file="MDS.pdf")
#cmd1=cmdscale(as.dist(distTOM2),4)
par(mfrow=c(2,3))
scatterplot3d(cmd1[,1:3], color=as.character(colorh1), main="MDS plot",xlab="Scaling
Dimension 1", ylab="Scaling Dimension 2", zlab="Scaling Dimension
3",cex.axis=0.5,cex.lab=0.5, angle=320)

scatterplot3d(cmd1[,2:4], color=as.character(colorh1), main="MDS plot",xlab="Scaling
Dimension 1", ylab="Scaling Dimension 2", zlab="Scaling Dimension 3",cex.axis=0.5,
cex.lab=0.5,angle=280)

scatterplot3d(cmd1[,1:3], color=as.character(colorh1), main="MDS plot",xlab="Scaling
Dimension 1", ylab="Scaling Dimension 2", zlab="Scaling Dimension 3",cex.axis=0.5,
cex.lab=0.5,angle=240)

scatterplot3d(cmd1[,2:4], color=as.character(colorh1), main="MDS plot",xlab="Scaling
Dimension 1", ylab="Scaling Dimension 2", zlab="Scaling Dimension 3",cex.axis=0.5,
cex.lab=0.5,angle=150)

scatterplot3d(cmd1[,1:3], color=as.character(colorh1), main="MDS plot",xlab="Scaling
Dimension 1", ylab="Scaling Dimension 2", zlab="Scaling Dimension 3",cex.axis=0.5,
cex.lab=0.5,angle=120)

scatterplot3d(cmd1[,2:4], color=as.character(colorh1), main="MDS plot",xlab="Scaling
Dimension 1", ylab="Scaling Dimension 2", zlab="Scaling Dimension 3",cex.axis=0.5,
cex.lab=0.5,angle=45)

dev.off()

# TOM plot
pdf(file="TOMplot_cut98_p6_6920_2703.pdf")
TOMplot1(distTOM2 , hierTOM2, colorh1)
dev.off()

#
pdf(file="Clustering_Module_Eigengenes(PC1)_cut98_2703.pdf")
PC1=ModulePrinComps1(datExpr2,colorh1)[[1]]
hclustPC1=hclust(as.dist( 1-abs(t(cor(PC1, method="p")))), method="average" )
par(mfrow=c(1,1))
plot(hclustPC1, main="Clustering the Module Eigengenes")

```

```
dev.off()
```

```
# Now we create scatter plots of the samples (arrays) along the module eigengenes.
```

```
pdf(file="Relationship_modules_cut98_p6_2703.pdf")
PC1=PC1[,hclustPC1$order]
pairs(PC1,upper.panel = panel.smooth, lower.panel = panel.cor, diag.panel=panel.hist
,main="Relation between modules")
dev.off()
```

```
# connectivity vs mean, sd
```

```
pdf(file="ScatterPlot_mediumseagreen_6920.pdf")
par(mfrow=c(2,2))
curr.power=6
beta=curr.power
DegreeSoft= SoftConnectivity(datExpr,power=beta)-1
Degree=DegreeSoft
whichmodule="mediumseagreen"
# mean expression of the blue module genes
meanExprModule=apply(datExpr2 [,colorh1==whichmodule],2,mean1)
# variance of expression
varExprModule=apply(datExpr2 [,colorh1==whichmodule],2,var1)
DegreeModule= SoftConnectivity(datExpr2[,colorh1==whichmodule], power=beta)
scatterplot1(DegreeModule,varExprModule,xlab1=paste("Connectivity (k.in)", whichmodule, "
module"), ylab1="Variance",coll=whichmodule)
scatterplot1(DegreeModule,meanExprModule,xlab1=paste("Connectivity (k.in)", whichmodule, "
module"), ylab1="Mean Expression",coll=whichmodule)
meanExpr=apply(datExpr,2,mean1)
varExpr=apply(datExpr,2,var1)
scatterplot1(Degree,varExpr,xlab1=paste("Whole Network Connectivity (k.all)"),
ylab1="Variance",coll=colorh1)
scatterplot1(Degree,meanExpr,xlab1=paste("Whole Network Connectivity (k.all)"),
ylab1="Mean Expression",coll=colorh1)
```

```
dev.off()
```

```
pdf(file="ScatterPlot_pink3.pdf")
par(mfrow=c(2,2))
whichmodule="pink3"
# mean expression of the blue module genes
meanExprModule=apply(datExpr2 [,colorh1==whichmodule],2,mean1)
# variance of expression
varExprModule=apply(datExpr2 [,colorh1==whichmodule],2,var1)
DegreeModule= SoftConnectivity(datExpr2[,colorh1==whichmodule], power=6)
scatterplot1(DegreeModule,varExprModule,xlab1=paste("Connectivity (k.in)", whichmodule, "
module"), ylab1="Variance",coll=whichmodule)
scatterplot1(DegreeModule,meanExprModule,xlab1=paste("Connectivity (k.in)", whichmodule, "
module"), ylab1="Mean Expression",coll=whichmodule)
meanExpr=apply(datExpr,2,mean1)
varExpr=apply(datExpr,2,var1)
scatterplot1(Degree,varExpr,xlab1=paste("Whole Network Connectivity (k.all)"),
ylab1="Variance",coll=colorh1)
scatterplot1(Degree,meanExpr,xlab1=paste("Whole Network Connectivity (k.all)"),
ylab1="Mean Expression",coll=colorh1)
```

```
dev.off()
```

appendix: R Functions

```
# This document contains functions that we find useful for gene co-expression network
analysis
# We recommend to read the tutorial first.
# Steve Horvath, Bin Zhang, Jun Dong, Andy Yip
# To cite this code or the statistical methods please use
# Zhang B, Horvath S (2005) A General Framework for Weighted Gene Co-Expression Network
Analysis.
# Statistical Applications in Genetics and Molecular Biology. In Press.
# Technical Report and software code at:
www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork.

# CONTENTS
# This document contains function for carrying out the following tasks
# A) Assessing scale free topology and choosing the parameters of the adjacency function
#   using the scale free topology criterion (Zhang and Horvath 05)
# B) Computing the topological overlap matrix
# C) Defining gene modules using clustering procedures
# D) Summing up modules by their first principal component (first eigengene)
# E) Relating a measure of gene significance to the modules
# F) Carrying out a within module analysis (computing intramodular connectivity)
#   and relating intramodular connectivity to gene significance.
# G) Miscellaneous other functions, e.g. for computing the cluster coefficient.
# H) Network functions from Bin Zhang for dynamic tree cutting of a hierarchical
clustering tree (dendrogram)
# I) General statistical functions, e.g. for scatterplots

# The functions below are organized according into categories A-G.
# One of these days, somebody should turn this into an R library...

#####
#####
#####
#####
# A) Assessing scale free topology and choosing the parameters of the adjacency function
#   using the scale free topology criterion (Zhang and Horvath 05)
#####
#####

# =====
#For hard thresholding, we use the signum (step) function
if(exists("signum1") ) rm(signum1);
signum1=function(corhelp,taul) {
adjmat1= as.matrix(abs(corhelp)>=taul)
dimnames(adjmat1) <- dimnames(corhelp)
diag(adjmat1) <- 0
adjmat1}

# =====
# For soft thresholding, one can use the sigmoid function
# But we have focused on the power adjacency function in the tutorial...
if (exists("sigmoid1") ) rm(sigmoid1); sigmoid1=function(ss,mu1=0.8,alpha1=20) {
1/(1+exp(-alpha1*(ss-mu1)))}
```

```

#This function is useful for speeding up the connectivity calculation.
#The idea is to partition the adjacency matrix into consecutive batches of a #given size.
#In principle, the larger the batch size the faster is the calculation. But #smaller
batchsizes require #less memory...
# Input: gene expression data set where *rows* correspond to microarray samples #and
columns correspond to genes.
# If fewer than MinimumNoSamples contain gene expression information for a given
# gene, then its connectivity is set to missing.
if(exists("SoftConnectivity")) rm(SoftConnectivity);
SoftConnectivity=function(datE, power=6,batchsize=1500,MinimumNoSamples=10) {
no.genes=dim(datE)[[2]]
no.samples=dim(datE)[[1]]
if (no.genes<no.samples | no.genes<10 | no.samples<5 ) {print("Error: Something seems to
be wrong. Make sure that the input data frame has genes as rows and array samples as
columns. Alternatively, there could be fewer than 10 genes or fewer than 5 samples. ") }
else {
suml=function(x) sum(x,na.rm=T)
k=rep(NA,no.genes)
no.batches=as.integer(no.genes/ batchsize)
if (no.batches>0) {
for (i in 1:no.batches) {
print(paste("batch number = ", i))
index1=c(1:batchsize)+(i-1)* batchsize
ad1=abs(cor(datE[,index1], datE,use="p"))^power
ad1[is.na(ad1)]=0
k[index1]=apply(ad1,1,suml)
# If fewer than MinimumNoSamples contain gene expression information for a given
# gene, then we set its connectivity to 0.
NoSamplesAvailable=apply(!is.na(datE[,index1]),2,sum)
k[index1][NoSamplesAvailable< MinimumNoSamples]=NA
} # end of for (i in 1:no.batches)
} # end of if (no.batches>0)...
if (no.genes-no.batches*batchsize>0 ) {
restindex=c((no.batches*batchsize+1):no.genes)
ad1=abs(cor(datE[,restindex], datE,use="p"))^power
ad1[is.na(ad1)]=0
k[restindex]=apply(ad1,1,suml)
NoSamplesAvailable=apply(!is.na(datE[,restindex]),2,sum)
k[restindex][NoSamplesAvailable< MinimumNoSamples]=NA
} # end of if
} # end of else statement
k
} # end of function

```

```

# =====
# The function PickHardThreshold can help one to estimate the cut-off value
# when using the signum (step) function.
# The first column lists the threshold ("cut"),
# the second column lists the corresponding p-value based on the Fisher Transform
# of the correlation.
# The third column reports the resulting scale free topology fitting index R^2.
# The fourth column reports the slope of the fitting line, it should be negative for
# biologically meaningful networks.
# The fifth column reports the fitting index for the truncated exponential model.
# Usually we ignore it.
# The remaining columns list the mean, median and maximum resulting connectivity.

```

```

# To pick a hard threshold (cut) with the scale free topology criterion:
# aim for high scale free R^2 (column 3), high connectivity (col 6) and negative slope
# (around -1, col 4).
# The output is a list with 2 components. The first component lists a suggested cut-off
# while the second component contains the whole table.
# The removeFirst option removes the first point (k=0, P(k=0)) from the regression fit.
# no.breaks specifies how many intervals used to estimate the frequency p(k) i.e. the no.
of points in the
# scale free topology plot.
if (exists("PickHardThreshold")) rm(PickHardThreshold);
PickHardThreshold=function(datExpr1, RsquaredCut=0.85, cutvector=seq(0.1,0.9,by=0.05)
,removeFirst=FALSE,no.breaks=10) {
no.genes <- dim(datExpr1)[[2]]
no.genes <- dim(datExpr1)[[2]]
no.samples= dim(datExpr1)[[1]]
colname1=c("Cut","p-value", "scale law R^2", "slope=" , "truncated
R^2","mean(k)","median(k)","max(k)")
datout=data.frame(matrix(666,nrow=length(cutvector),ncol=length(colname1) ))
names(datout)=colname1
datout[,1]=cutvector
for (i in c(1:length(cutvector) ) ){
cut1=cutvector[i]
datout[i,2]=2*(1-pt(sqrt(no.samples-1)*cut1/sqrt(1-cut1^2),no.samples-1))}
if(exists("fun1")) rm(fun1)
fun1=function(x) {
corx=abs(cor(x,datExpr1,use="p"))
out1=rep(NA, length(cutvector) )
for (j in c(1:length(cutvector))) {out1[j]=sum(corx>cutvector[j])}
out1
} # end of fun1
datk=t(apply(datExpr1,2,fun1))
for (i in c(1:length(cutvector) ) ){
nolinkshelp <- datk[,i]-1
cut2=cut(nolinkshelp,no.breaks)
binned.k=tapply(nolinkshelp,cut2,mean)
freq1=as.vector(tapply(nolinkshelp,cut2,length)/length(nolinkshelp))
# The following code corrects for missing values etc
breaks1=seq(from=min(nolinkshelp),to=max(nolinkshelp),length=no.breaks+1)
hist1=hist(nolinkshelp,breaks=breaks1,equidist=F,plot=FALSE,right=TRUE)
binned.k2=hist1$mids
binned.k=ifelse(is.na(binned.k),binned.k2,binned.k)
binned.k=ifelse(binned.k==0,binned.k2,binned.k)
freq1=ifelse(is.na(freq1),0,freq1)
xx= as.vector(log10(binned.k))
if(removeFirst) {freq1=freq1[-1]; xx=xx[-1]}
plot(xx,log10(freq1+.000000001),xlab="log10(k)",ylab="log10(p(k))" )
lm1= lm(as.numeric(log10(freq1+.000000001))~ xx )
lm2=lm(as.numeric(log10(freq1+.000000001))~ xx+I(10^xx) )
datout[i,3]=summary(lm1)$adj.r.squared
datout[i,4]=summary(lm1)$coefficients[2,1]
datout[i,5]=summary(lm2)$adj.r.squared
datout[i,6]=mean(nolinkshelp)
datout[i,7]= median(nolinkshelp)
datout[i,8]= max(nolinkshelp)
}
datout=signif(datout,3)
print(data.frame(datout));
# the cut-off is chosen as smallest cut with R^2>RsquaredCut
ind1=datout[,3]>RsquaredCut
indcut=NA
indcut=ifelse(sum(ind1)>0,min(c(1:length(ind1))[ind1]),indcut)

```

```

# this estimates the cut-off value that should be used.
# Don't trust it. You need to consider slope and mean connectivity as well!
cut.estimate=cutvector[indcut][[1]]
list(cut.estimate, data.frame(datout));
} # end of function

```

```

# =====
# The function PickSoftThreshold allows one to estimate the power parameter when using
# a soft thresholding approach with the use of the power function  $AF(s)=s^{Power}$ 
# The function PickSoftThreshold allows one to estimate the power parameter when using
# a soft thresholding approach with the use of the power function  $AF(s)=s^{Power}$ 
# The removeFirst option removes the first point ( $k=1$ ,  $P(k=1)$ ) from the regression fit.
if (exists("PickSoftThreshold")) rm(PickSoftThreshold);
PickSoftThreshold=function(datExpr1, RsquaredCut=0.85,
powervector=c(seq(1,10,by=1), seq(12,20,by=2)),
removeFirst=FALSE, no.breaks=10) {
no.genes <- dim(datExpr1)[[2]]
no.genes <- dim(datExpr1)[[2]]
no.samples= dim(datExpr1)[[1]]
colname1=c("Power", "scale law  $R^2$ ", "slope", "truncated
 $R^2$ ", "mean(k)", "median(k)", "max(k)")
datout=data.frame(matrix(666, nrow=length(powervector), ncol=length(colname1) ))
names(datout)=colname1
datout[,1]=powervector
if(exists("fun1")) rm(fun1)
fun1=function(x) {
corx=abs(cor(x, datExpr1, use="p"))
out1=rep(NA, length(powervector) )
for (j in c(1:length(powervector))) {out1[j]=sum(corx^powervector[j])}
out1
} # end of fun1
datk=t(apply(datExpr1, 2, fun1))
for (i in c(1:length(powervector) )){
nolinkshelp <- datk[,i]-1
cut2=cut(nolinkshelp, no.breaks)
binned.k=tapply(nolinkshelp, cut2, mean)
freq1=as.vector(tapply(nolinkshelp, cut2, length)/length(nolinkshelp))
# The following code corrects for missing values etc
breaks1=seq(from=min(nolinkshelp), to=max(nolinkshelp), length=no.breaks+1)
hist1=hist(nolinkshelp, breaks=breaks1, equidist=F, plot=FALSE, right=TRUE)
binned.k2=hist1$mids
binned.k=ifelse(is.na(binned.k), binned.k2, binned.k)
binned.k=ifelse(binned.k==0, binned.k2, binned.k)
freq1=ifelse(is.na(freq1), 0, freq1)

xx= as.vector(log10(binned.k))
if(removeFirst) {freq1=freq1[-1]; xx=xx[-1]}
plot(xx, log10(freq1+.000000001), xlab="log10(k)", ylab="log10(p(k))" )
lm1= lm(as.numeric(log10(freq1+.000000001))~ xx )
lm2=lm(as.numeric(log10(freq1+.000000001))~ xx+I(10^xx) )
datout[i, 2]=summary(lm1)$adj.r.squared

```

```

datout[i,3]=summary(lm1)$coefficients[2,1]
datout[i,4]=summary(lm2)$adj.r.squared
datout[i,5]=mean(nolinkshelp)
datout[i,6]= median(nolinkshelp)
datout[i,7]= max(nolinkshelp)
}
datout=signif(datout,3)
print(data.frame(datout));
# the cut-off is chosen as smallest cut with R^2>RsquaredCut
ind1=datout[,2]>RsquaredCut
indcut=NA
indcut=ifelse(sum(ind1)>0,min(c(1:length(ind1))[ind1]),indcut)
# this estimates the power value that should be used.
# Don't trust it. You need to consider slope and mean connectivity as well!
power.estimate=powervector[indcut][[1]]
list(power.estimate, data.frame(datout));
}

# =====
# The function ScaleFreePlot1 creates a plot for checking scale free topology
# when truncated1=T is specified, it provides the R^2 measures for the following
# degree distributions: a) scale free topology, b) log-log R^2 and c) truncated
exponential R^2

# The function ScaleFreePlot1 creates a plot for checking scale free topology
if(exists("ScaleFreePlot1")) rm(ScaleFreePlot1) ;
ScaleFreePlot1=function(kk,no.breaks=10,AF1="" ,truncated1=FALSE,
removeFirst=FALSE,cex.lab1=1){
cut1=cut(kk,no.breaks)
binned.k=tapply(kk,cut1,mean)
freq1=tapply(kk,cut1,length)/length(kk)
# The following code corrects for missing values etc
breaks1=seq(from=min(kk),to=max(kk),length=no.breaks+1)
hist1=hist(kk,breaks=breaks1,equidist=F,plot=FALSE,right=TRUE)
binned.k2=hist1$mids
binned.k=ifelse(is.na(binned.k),binned.k2,binned.k)
binned.k=ifelse(binned.k==0,binned.k2,binned.k)
freq1=ifelse(is.na(freq1),0,freq1)
plot(log10(binned.k),log10(freq1+.00000001),xlab="log10(k)",ylab="log10(p(k))",cex.lab=c
ex.lab1 )
xx= as.vector(log10(binned.k))
if(removeFirst) {freq1=freq1[-1]; xx=xx[-1]}
lm1=lm(as.numeric(log10(freq1+.00000001))~ xx )
lines(xx,predict(lm1),col=1)
OUTPUT=data.frame(ScaleFreeRsquared=round(summary(lm1)$adj.r.squared,2),Slope=round(lm1$c
oefficients[[2]],2))
if (truncated1==TRUE) {
lm2=lm(as.numeric(log10(freq1+.00000001))~ xx+I(10^xx) );
OUTPUT=data.frame(ScaleFreeRsquared=round(summary(lm1)$adj.r.squared,2),Slope=round(lm1$c
oefficients[[2]],2),
TruncatedRsquared=round(summary(lm2)$adj.r.squared,2))
print("the red line corresponds to the truncated exponential fit")
lines(xx,predict(lm2),col=2);
title(paste(AF1,
", scale free R^2=",as.character(round(summary(lm1)$adj.r.squared,2)),
", slope=", round(lm1$coefficients[[2]],2),

```

```

", trunc.R^2=",as.character(round(summary(lm2)$adj.r.squared,2))))} else {
title(paste(AF1, ", scale R^2=",as.character(round(summary(lm1)$adj.r.squared,2)) ,
", slope=", round(lm1$coefficients[[2]],2)))
}
OUTPUT
} # end of function

```

```

#####
#####
#####
#####
# B) Computing the topological overlap matrix
#####
#####
#####
#####

```

```

# =====
#The function TOMdist1 computes a dissimilarity
# based on the topological overlap matrix (Ravasz et al)
# Input: an Adjacency matrix with entries in [0,1]
if(exists("TOMdist1")) rm(TOMdist1);
TOMdist1=function(adjmat1, maxADJ=FALSE) {
diag(adjmat1)=0;
adjmat1[is.na(adjmat1)]=0;
maxh1=max(as.dist(adjmat1) ); minh1=min(as.dist(adjmat1) );
if (maxh1>1 | minh1 < 0 ) {print(paste("ERROR: the adjacency matrix contains entries that
are larger than 1 or smaller than 0!!!, max=",maxh1,", min=",minh1)) } else {
if ( max(c(as.dist(abs(adjmat1-t(adjmat1))))))>0 ) {print("ERROR: non-symmetric
adjacency matrix!!!") } else {
kk=apply(adjmat1,2,sum)
maxADJconst=1
if (maxADJ==TRUE) maxADJconst=max(c(as.dist(adjmat1 )))
Dhelp1=matrix(kk,ncol=length(kk),nrow=length(kk))
denomTOM= pmin(as.dist(Dhelp1),as.dist(t(Dhelp1))) +as.dist(maxADJconst-adjmat1);
gc();gc();
numTOM=as.dist(adjmat1 %*% adjmat1 +adjmat1);
#TOMmatrix=numTOM/denomTOM
# this turns the TOM matrix into a dissimilarity
out1=1-as.matrix(numTOM/denomTOM)
diag(out1)=1
out1
}}
}

```

```

# =====
# This function computes a TOMk dissimilarity
# which generalizes the topological overlap matrix (Ravasz et al)
# Input: an Adjacency matrix with entries in [0,1]

```

```

# WARNING: ONLY FOR UNWEIGHTED NETWORKS, i.e. the adjacency matrix contains binary
entries...
# This function is explained in Yip and Horvath (2005)
# http://www.genetics.ucla.edu/labs/horvath/GTOM/
if(exists("TOMkdist1")) rm(TOMkdist1);
TOMkdist1 = function(adjmat1,k=1){
  maxh1=max(as.dist(adjmat1) ); minh1=min(as.dist(adjmat1) );
  if (k!=round(abs(k))) {
    stop("k must be a positive integer!!!", call.=TRUE);}
  if (maxh1>1 | minh1 < 0 ){
    print(paste("ERROR: entries of the adjacency matrix must be between inclusively 0
and 1!!!, max=",maxh1,", min=",minh1))}
  else {
if ( max(c(as.dist(abs(adjmat1-t(adjmat1))))>0 ) {print("ERROR: non-symmetric
adjacency matrix!!!") } else {

  B <- adjmat1;
  if (k>=2) {
    for (i in 2:k) {
      diag(B) <- diag(B) + 1;
      B = B %%% adjmat1;}} # this gives the number of paths with length at
most k connecting a pair
  B <- (B>0); # this gives the k-step reachability from a node to another
  diag(B) <- 0; # exclude each node being its own neighbor
  B <- B %%% B # this gives the number of common k-step-neighbor that a pair of
nodes share

  Nk <- diag(B);
  B <- B +adjmat1; # numerator
  diag(B) <- 1;
  denomTOM=outer(Nk,Nk,FUN="pmin")+1-adjmat1;
  diag(denomTOM) <- 1;
  1 - B/denomTOM # this turns the TOM matrix into a dissimilarity
}}
}

# IGNORE THIS function...
# The function TOMdistROW computes the TOM distance of a gene (node)
# with that of all other genes in the network.
# WhichRow is an integer that specifies which row of the adjacency matrix
# corresponds to the gene of interest.
# Output=vector of TOM distances.
if (exists("TOMdistROW") ) rm(TOMdistROW)
TOMdistROW=function(WhichRow=1, adjmat1, maxADJ=FALSE) {
diag(adjmat1)=0;
maxh1=max(as.dist(adjmat1) ); minh1=min(as.dist(adjmat1) );
if (maxh1>1 | minh1 < 0 ) {print(paste("ERROR: the adjacency matrix contains entries that
are larger than 1 or smaller than 0!!!, max=",maxh1,", min=",minh1)) } else {
kk=apply(adjmat1,2,sum)
numTOM=adjmat1[WhichRow,] %%% adjmat1 +adjmat1[WhichRow,];
numTOM[WhichRow]=1
maxADJconst=1
if (maxADJ==TRUE) maxADJconst=max(c(as.dist(adjmat1) ))
denomTOM=pmin(kk[WhichRow],kk)+maxADJconst-adjmat1[WhichRow,]; denomTOM[WhichRow]=1
#TOMmatrix=numTOM/denomTOM
# this turns the TOM matrix into a dissimilarity
1-numTOM/denomTOM
}
}
}

```

```

#####
#####
#####
#####
# C) Defining gene modules using clustering procedures
#####
#####
#####

# =====
#The function modulecolor2 function assigns colors to the observations
# in the branches of a dendrogram
# we use it to define modules....
if (exists("modulecolor2")) rm(modulecolor2);
modulecolor2=function(hier1, h1=0.9, minsize1=50) {
# here we define modules by using a height cut-off for the branches
labelpred= cutree(hier1, h=h1)
sort1=-sort(-table(labelpred))
modulename= as.numeric(names(sort1))
modulebranch= sort1>minsize1
no.modules=sum(modulebranch)
# now we assume that there are fewer than a certain number of colors
colorcode=c("turquoise", "blue", "brown", "yellow", "green", "red", "black", "purple", "orange", "
pink",
"greenyellow", "lightcyan", "salmon", "midnightblue", "lightyellow")
# "grey" means not in any module;
colorhelp=rep("grey", length(labelpred))
if ( no.modules==0 | no.modules >length(colorcode)){ print(paste("The number of modules
is problematic! \! Number of modules = ", as.character(no.modules)))} else { for (i in
c(1:no.modules)) {colorhelp=ifelse(labelpred==modulename[i], colorcode[i], colorhelp)};
colorhelp=factor(colorhelp, levels=c(colorcode[1:no.modules], "grey"))
}
factor(colorhelp, levels=unique(colorhelp[hier1$order] ))
}

# The function hclustplot1 creates a barplot where the colors of the bars are sorted
according to
# a hierarchical clustering tree (hclust object)
if (exists("hclustplot1")) rm(hclustplot1);
hclustplot1=function(hier1, couleur, title1="Colors sorted by hierarchical clustering") {
if (length(hier1$order) != length(couleur) ) {print("ERROR: length of color vector not
compatible with no. of objects in the hierarchical tree")};
if (length(hier1$order) == length(couleur) ) {
barplot(height=rep(1, length(couleur)), col= as.character(couleur[hier1$order]), border=F,
main=title1, space=0, axes=F)}
}

# =====
# The function TOMplot1 creates a TOM plot
# Inputs: distance measure, hierarchical (hclust) object, color label=couleur

```

```

if (exists("TOMplot1")) rm(TOMplot1);
TOMplot1=function(disttom,hier1, couleur,terrainColors=FALSE) {
no.nodes=length(couleur)
if (no.nodes != dim(disttom)[[1]] ) {print("ERROR: number of color labels does not equal
number of nodes in disttom")} else {
labeltree=as.character(couleur)
labelrow = labeltree
labelrow[hier1$order[length(labeltree):1]]=labelrow[hier1$order]
options(expressions = 10000)
if (terrainColors) heatmap(as.matrix(disttom),Rowv=as.dendrogram(hier1),Colv=
as.dendrogram(hier1), scale="none",revC=T,
ColSideColors=as.character(labeltree),RowSideColors=as.character(labelrow),
labRow=F, labCol=F, col = terrain.colors(1000)) else
heatmap(as.matrix(disttom),Rowv=as.dendrogram(hier1),Colv= as.dendrogram(hier1),
scale="none",revC=T,
ColSideColors=as.character(labeltree),RowSideColors=as.character(labelrow),
labRow=F, labCol=F)
}
} #end of function

```

```

# =====
# The function TOMplot2 creates a TOM plot where the top and left color bars can be
different
# Inputs: distance measure, hierarchical (hclust) object, color label=couleurTop,
couleurLeft
if (exists("TOMplot2")) rm(TOMplot2);
TOMplot2=function(disttom,hier1, couleurTop, couleurLeft) {
no.nodes=length(couleurTop)
if (no.nodes != length(couleurLeft)) {stop("ERROR: number of top color labels does not
equal number of left color labels")}
if (no.nodes != dim(disttom)[[1]] ) {stop("ERROR: number of color labels does not equal
number of nodes in disttom")} else {
labeltree = as.character(couleurTop)
labelrow = as.character(couleurLeft)
labelrow[hier1$order[length(labeltree):1]]=labelrow[hier1$order]
options(expressions = 10000)
heatmap(as.matrix(disttom),Rowv=as.dendrogram(hier1),Colv= as.dendrogram(hier1),
scale="none", revC=T,
ColSideColors=as.character(labeltree),RowSideColors=as.character(labelrow),
labRow=F, labCol=F)
}
} #end of function

```

```

# IGNORE THIS FUNCTION...
# The function îBestHeightCutî allows one to find the best height cut-off
# for a hierarchical clustering tree when external gene information is available
# It computes a Kruskal Wallis-test p-value for each height cut-off
# based on determining whether gene significance differs across branch membership.
if(exists("BestHeightCut")) rm(BestHeightCut);
BestHeightCut=function(hier1, GeneSignif, hcut=seq(0.1,.95,by=0.01) ) {
pvalues=rep(NA, length(hcut))
for (i in c(1:length(hcut))) {
colorhelp=modulecolor2(hier1,hcut[i])
if (length(unique(colorhelp))>1 ) {pvalues[i]=kruskal.test(GeneSignif,
colorhelp)$p.value}
data.frame(hcut,pvalues)
}}

```

```

#####
#####
#####
#####
# D) Summing up modules using their first principal components (first eigengene)
#####
#####
#####
#####

# =====
#The function ModulePrinComps1 finds the first principal component (eigengene) in each
# module defined by the colors of the input vector "couleur" (Pardon my French).
# It also reports the variances explained by the first 5 principal components.
# And it yields a measure of module conformity for each gene,
# which is highly correlated to the within module connectivity.
# The theoretical underpinnings are described in Horvath, Dong, Yip (2005)
# http://www.genetics.ucla.edu/labs/horvath/ModuleConformity/
# This requires the R library impute
# The output is a list with 3 components:
# 1) a data frame of module eigengenes (PCs),
# 2) a data frame that lists the percent variance explained by the first 5 PCs of a
module
# 3) a data frame that lists the module conformity for each gene.
# The be used as alternative connectivity measure....
if(exists("ModulePrinComps1")) rm(ModulePrinComps1);
ModulePrinComps1=function(datexpr,couleur) {
  modlevels=levels(factor(couleur))
  PrinComps=data.frame(matrix(666,nrow=dim(datexpr)[[1]],ncol= length(modlevels)))
  varexplained= data.frame(matrix(666,nrow= 5,ncol= length(modlevels)))
  names(PrinComps)=paste("PC",modlevels,sep="")
  for(i in c(1:length(modlevels)) ){
    print(i)
    modulename      = modlevels[i]
    restrict1= as.character(couleur)== modulename
    # in the following, rows are genes and columns are samples
    datModule=t(datexpr[, restrict1])
    datModule=impute.knn(as.matrix(datModule))
  datModule=t(scale(t(datModule)))
  svd1=svd(datModule)
    mtitle=paste("PCs of ", modulename," module", sep="")
    varexplained[,i]= (svd1$d[1:5])^2/sum(svd1$d^2)
  # this is the first principal component
    pc1=svd1$v[,1]
    signh1=sign(sum(cor(pc1, t(datModule))))
    if (signh1 != 0) pc1=signh1* pc1
  PrinComps[,i]= pc1
  }
  ModuleConformity= rep(666,length=dim(datexpr)[[2]])
  for(i in 1:(dim(datexpr)[[2]])) ModuleConformity[i]=abs(cor(datexpr[,i],
  PrinComps[,match(couleur[i], modlevels)], use="pairwise.complete.obs"))
  list(PrinComps=PrinComps, varexplained=varexplained, ModuleConformity=ModuleConformity)
}

#####
#####

```

```

#####
#####
# E) Relating a measure of gene significance to the modules
#####
#####
#####

# =====
# The function ModuleEnrichment1 creates a bar plot that shows whether modules are
enriched with
# significant genes.
# More specifically, it reports the mean gene significance for each module.
# The gene significance can be a binary variable or a quantitative variable.
# It also plots the 95% confidence interval of the mean (CI=mean +/- 1.96* standard
error).
# It also reports a Kruskal Wallis P-value.
if( exists("ModuleEnrichment1") ) rm(ModuleEnrichment1);
ModuleEnrichment1=function(genesignif1,couleur,title1="gene significance across
modules",labely="Gene Significance",boxplot=F) {
if (length(genesignif1) != length(couleur) ) print("Error: vectors don't have the same
lengths") else {
if (boxplot != TRUE) {
mean1=function(x) mean(x,na.rm=T)
means1=as.vector(tapply(genesignif1,couleur,mean1));
sel= as.vector(tapply(genesignif1,couleur,stderr1))
par(mfrow=c(1,1))
barplot(means1,
names.arg=names(table(couleur) ),col= names(table(couleur) )
,ylab=labely)
err.bp(as.vector(means1), as.vector(1.96*sel), two.side=T)} else {
boxplot(split(genesignif1,couleur),notch=T,varwidth=T, col= names(table(couleur)
),ylab=labely)}

title(paste(title1,", p-value=",
signif(kruskal.test(genesignif1,factor(couleur))$p.value,2))
)
} # end of function

# IGNORE THIS...
# =====
#The function fisherPvector allows one to compute Fisher exact p-values
# Thus it allows one to carry out an EASE analysis
# Output: a table of Fisher's exact p-values
# Input: annotation1 is a vector of gene annotations
# Input: couleur (French for color) denotes the module color of each gene
# Only those gene functions (levels of annotation1) that occur a certain minimum number
of times
# (parameter= minNumberAnnotation) in the data set will be considered.
if (exists("fisherPvector" ) ) rm(fisherPvector);
fisherPvector=function(couleur,annotation1,minNumberAnnotation=50) {
levelsannotation1=levels(annotation1)
levelscouleur=levels(factor(couleur))
no.couleur=length(levelscouleur)
restannotation1=table(annotation1)>minNumberAnnotation
no.annotation=sum( restannotation1)
datoutP=data.frame(matrix(666,nrow=no.annotation,ncol=no.couleur) )
#datoutProp=data.frame(matrix(666,nrow=no.annotation,ncol=2*no.couleur) )
#names(datoutProp)=paste("Prop",paste( rep(levelscouleur ,rep(2, length(levelscouleur)))
) , c("Y","N") ,sep=".")

```



```

WithinModuleAnalysis1=function(datnetwork,nodesignif, couleur) {
cortesthelp=function( x ) {
len1=dim(x)[[2]]-1
out1=rep(666, len1);
for ( i in c(1:len1) ) {out1[i]= signif( cor.test(x[,i+1], x[,1], method="s",use="p"
)$p.value ,2) }
data.frame( variable=names(x)[-1] , NS.CorPval=out1, NS.cor=t(signif(cor (x[,1], x[,-
1],use="p",method="s"),2)), signif(cor(x[,-1],use="p",method="s"),2) )
} #end of function cortesthelp
print("IGNORE the warnings...");
by( data.frame(nodesignif, datnetwork),couleur,cortesthelp);
} #end of function WithinModuleAnalysis

# =====
# The function WithinModuleCindex1 relates the node measures (e.g. connectivities)
# to "external" node significance information within each module,
# i.e. it carries out a by module analysis.
# BUT it focuses on the C-index also known as area under the ROC curve
# This measure is related to Kendall's Tau statistic and Somer's D,
# see F. Harrel (Regression Modeling Strategies). Springer.
# It requires the following library
library(Hmisc)
# Output: the first column reports the C-index and the second, p-value
if (exists("WithinModuleCindex1")) rm(WithinModuleCindex1);
WithinModuleCindex1=function(datnetwork,nodesignif, couleur) {
CindexFun=function( x ) {
len1=dim(x)[[2]]-1
outC=rep(666, len1);
outP=rep(666, len1);
for ( i in c(1:len1) ) {rcor1=rcorr.cens(x[,i+1], x[,1])
outC[i]=rcor1[[1]]
outP[i]=1- pnorm(abs(rcor1[[2]]/rcor1[[3]]))
}
data.frame( variable=names(x)[-1] , C.index=outC, p.value=outP)
} #end of function CindexFun
#print("IGNORE the warnings...");
by( data.frame(nodesignif, datnetwork),couleur,CindexFun);
} #end of function WithinModuleAnalysis

# The following function allows on to plot a gene (node) significance measure versus
# connectivity.
if(exists("plotConnectivityGeneSignif1") ) rm( plotConnectivityGeneSignif1);
plotConnectivityGeneSignif1=function(degreel,genesignif1,color1="black",
title1="Gene Significance vs Connectivity" , xlab1="Connectivity",
ylab1="GeneSignificance") {
lm1=lm(genesignif1~degreel ,na.action="na.omit")
plot(degreel, genesignif1, col=color1,ylab=ylab1,xlab=xlab1,main=paste(title1, ", cor=",
signif(cor( genesignif1,degreel, method="s",use="p" ) ,2) ))
abline(lm1)
}

#####
#####

```

```
#####
#####
# G) Miscellaneous other functions, e.g. for computing the cluster coefficient.
#####
#####
#####
#####
```

```
# =====
# The function ClusterCoef.fun computes the cluster coefficients.
# Input is an adjacency matrix
if(exists("ClusterCoef.fun")) rm(ClusterCoef.fun) ; ClusterCoef.fun=function(adjmat1) {
diag(adjmat1)=0
no.nodes=dim(adjmat1)[[1]]
computeLinksInNeighbors <- function(x, imatrix){x %*% imatrix %*% x}
nolinksNeighbors <- c(rep(-666,no.nodes))
total.edge <- c(rep(-666,no.nodes))
maxhl=max(as.dist(adjmat1) ); minhl=min(as.dist(adjmat1) );
if (maxhl>1 | minhl < 0 ) {print(paste("ERROR: the adjacency matrix contains entries that
are larger than 1 or smaller than 0!!!, max=",maxhl,", min=",minhl)) } else {
nolinksNeighbors <- apply(adjmat1, 1, computeLinksInNeighbors, imatrix=adjmat1)
plainsum <- apply(adjmat1, 1, sum)
squaresum <- apply(adjmat1^2, 1, sum)
total.edge = plainsum^2 - squaresum
CChelp=rep(-666, no.nodes)
CChelp=ifelse(total.edge==0,0, nolinksNeighbors/total.edge)
CChelp}
} # end of function
```

```
# =====
# The function err.bp is used to create error bars in a barplot
# usage: err.bp(as.vector(means), as.vector(stderrs), two.side=F)
err.bp<-function(daten,error,two.side=F){
  if(!is.numeric(daten)) {
    stop("All arguments must be numeric")}
  if(is.vector(daten)){
    xval<-(cumsum(c(0.7,rep(1.2,length(daten)-1))))
  }else{
    if (is.matrix(daten)){
      xval<-cumsum(array(c(1,rep(0,dim(daten)[1]-1)),
dim=c(1,length(daten)))+0:(length(daten)-1)+.5
    )}else{
      stop("First argument must either be a vector or a matrix") }
  }
  MW<-0.25*(max(xval)/length(xval))
  ERR1<-daten+error
  ERR2<-daten-error
  for(i in 1:length(daten)){
    segments(xval[i],daten[i],xval[i],ERR1[i])
    segments(xval[i]-MW,ERR1[i],xval[i]+MW,ERR1[i])
    if(two.side){
      segments(xval[i],daten[i],xval[i],ERR2[i])
      segments(xval[i]-MW,ERR2[i],xval[i]+MW,ERR2[i])
    }
  }
}
```

```

# =====
# this function computes the standard error
if (exists("stderr1")) rm(stderr1)
stderr1 <- function(x){ sqrt( var(x,na.rm=T)/sum(!is.na(x)) ) }

# =====
# The following two functions are for displaying the pair-wise correlation in a panel
when using the command "pairs()"
# Typically, we use "pairs(DATA, upper.panel=panel.smooth, lower.panel=panel.cor,
diag.panel=panel.hist)" to
# put the correlation coefficients on the lower panel.
panel.cor <- function(x, y, digits=2, prefix="", cex.cor){
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))
  txt <- format(c(r, 0.123456789), digits=digits)[1]
  txt <- paste(prefix, txt, sep="")
  if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
  text(0.5, 0.5, txt, cex = cex * r)
}
panel.hist <- function(x, ...){
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col="cyan", ...)
}

# =====
# this function computes the standard error
if (exists("stderr1")) rm(stderr1);
stderr1 <- function(x){ sqrt( var(x,na.rm=T)/sum(!is.na(x)) ) }

# =====
# This function collects garbage
if (exists("collect_garbage")) rm(collect_garbage);
collect_garbage=function(){while (gc()[2,4] != gc()[2,4] | gc()[1,4] != gc()[1,4]){} }
collect_garbage()

# this function is used for computing the Rand index below...
# =====
if (exists("choosenew") ) rm(choosenew)
choosenew <- function(n,k){
  n <- c(n)
  out1 <- rep(0,length(n))
  for (i in c(1:length(n)) ){
    if (n[i]<k) {out1[i] <- 0}
    else {out1[i] <- choose(n[i], k)}}
  out1
}

```

```

# =====
# the following function computes the Rand index between 2 clusterings
# assesses how similar two clusterings are
if (exists("Rand1") ) rm(Rand1)
Rand2 <- function(tab,adjust=T) {
  a <- 0; b <- 0; c <- 0; d <- 0; nn <- 0
  m <- nrow(tab);
  n <- ncol(tab);
  for (i in 1:m) {
    c<-0
    for(j in 1:n) {
      a <- a+choosenew(tab[i,j],2)
      nj <- sum(tab[,j])
      c <- c+choosenew(nj,2)
    }
    ni <- sum(tab[i,])
    b <- b+choosenew(ni,2)
    nn <- nn+ni
  }
  if(adjust==T) {
    d <- choosenew(nn,2)
    adrand <- (a-(b*c)/d)/((0.5*(b+c)-(b*c)/d)
    adrand
  } else {
    b <- b-a
    c <- c-a
    d <- choosenew(nn,2)-a-b-c
    rand <- (a+d)/(a+b+c+d)
    rand
  }
}

# =====
# This function is used in "pairs()" function. The problem of the original panel.cor is
that
# when the correlation coefficient is very small, the lower panel will have a large font
# instead of a mini-font in a saved .ps file. This new function uses a format for
corr=0.2
# when corr<0.2, but it still reports the original value of corr, with a minimum format.

panel.cor1=function(x, y, digits=2, prefix="", cex.cor){
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))
  txt <- format(c(r, 0.123456789), digits=digits)[1]
  txt <- paste(prefix, txt, sep="")
  txt1=txt
  r1=r
  if (r<0.2) {
    r1=0.2
    txt1 <- format(c(r1, 0.123456789), digits=digits)[1]
    txt1 <- paste(prefix, txt1, sep="")
  }
  if(missing(cex.cor)) cex <- 0.8/strwidth(txt1)
  cex = cex * r1
  r <- round(r, digits)
  txt <- format(c(r, 0.123456789), digits=digits)[1]
  txt <- paste(prefix, txt, sep="")
  text(0.5, 0.5, txt, cex=cex)
}

```

```
# H) THE FOLLOWING FUNCTIONS were created by Bin Zhang. It allows one to cut small
branches of a hierarchical tree.
# For a description see http://www.genetics.ucla.edu/labs/horvath/binzhang/DynamicTreeCut
```

```
if( exists("cutreeDynamic") ) rm(cutreeDynamic)

cutreeDynamic = function(hierclust, maxTreeHeight=1, deepSplit=TRUE, minModuleSize=50,
minAttachModuleSize=100, nameallmodules=FALSE, useblackwhite=FALSE)
{
  if(maxTreeHeight >=1){
    staticCutCluster = cutTreeStatic(hiercluster=hierclust, heightcutoff=0.99,
minsize1=minModuleSize)
  }else{
    staticCutCluster = cutTreeStatic(hiercluster=hierclust, heightcutoff=maxTreeHeight,
minsize1=minModuleSize)
  }

  #get tree height for every singleton
  #node_index tree_height
  demdroHeiAll= rbind( cbind(hierclust$merge[,1], hierclust$height),
cbind(hierclust$merge[,2], hierclust$height) )

  #singletons will stand at the front of the list
  myorder = order(demdroHeiAll[,1])

  #get # of singletons
  no.singletons = length(hierclust$order)
  demdroHeiAll.sort = demdroHeiAll[myorder, ]
  demdroHei.sort = demdroHeiAll.sort[c(1:no.singletons), ]
  demdroHei = demdroHei.sort[seq(no.singletons, 1, by=-1), ]
  demdroHei[,1] = -demdroHei[,1]

  # combine with preliminary cluster-cutoff results
  demdroHei = cbind(demdroHei, as.integer(staticCutCluster))

  # re-order the order based on the dendrogram order hierclust$order
  demdroHei.order = demdroHei[hierclust$order, ]

  static.clupos = locateCluster(demdroHei.order[, 3])
  static.no = dim(static.clupos)[1]

  static.clupos2 = static.clupos
  static.no2 = static.no

  #split individual cluster if there are sub clusters embedded
  mcycle=1
  while(1!=1){
    clupos = NULL
```

```

    for (i in c(1:static.no)){
        mydemdroHei.order = demdroHei.order[ c(static.clupos[i,1]:static.clupos[i,2]),
] #index to [1, clusterSize]
        mydemdroHei.order[, 1] = mydemdroHei.order[, 1] - static.clupos[i, 1] + 1

        #cat("Cycle ", as.character(mcycle), "cluster (", static.clupos[i,1],
static.clupos[i,2], ")\n")
        #cat("i=", as.character(i), "\n")

        iclupos = processInvididualCluster(mydemdroHei.order,
                                           cminModuleSize = minModuleSize,
                                           cminAttachModuleSize = minAttachModuleSize)

        iclupos[,1] = iclupos[,1] + static.clupos[i, 1] -1 #recover the original index
        iclupos[,2] = iclupos[,2] + static.clupos[i, 1] -1

        clupos = rbind(clupos, iclupos) #put in the final output buffer
    }

    if(deepSplit==FALSE){
        break
    }

    if(dim(clupos)[1] != static.no) {
        static.clupos = clupos
        static.no = dim(static.clupos)[1]
    }else{
        break
    }
}
mcycle = mcycle + 1
#static.clupos
}
final.cnt = dim(clupos)[1]
#assign colors for modules
module.assign = rep(0, no.singletons)
module.cnt=1
for (i in c(1:final.cnt ))
{
    sdx = clupos[i, 1] #module start point
    edx = clupos[i, 2] #module end point
    module.size = edx - sdx +1
    if(module.size <minModuleSize){
        next
    }
    #assign module lable
    module.assign[sdx:edx] = rep(module.cnt, module.size)
    #update module label for the next module
    module.cnt = module.cnt + 1
}
colcode.reduced.order = assignModuleColor(module.assign, minsize1=minModuleSize,
anameallmodules=nameallmodules, auseblackwhite=useblackwhite)
recov.order = order( demdroHei.order[,1])
colcode.reduced = colcode.reduced.order[recov.order]
if(1==2){
    aveheight = averageSequence(demdroHei.order[,2], 2)
    procHei = demdroHei.order[,2]-mean(demdroHei.order[,2])
    par(mfrow=c(3,1), mar=c(0,0,0,0) )
    plot(hlrow, labels=F, xlab="",ylab="",main="",sub="",axes = F)
    barplot(procHei,
            col= "black", space=0,
            border=F,main="", axes = F, axisnames = F)
}

```

```

        barplot(height=rep(1, length(colcode.reduced)),
                col= as.character(colcode.reduced[h1row$order]), space=0,
                border=F,main="", axes = F, axisnames = F)
    par(mfrow=c(1,1), mar=c(5, 4, 4, 2) + 0.1)
}

colcode.reduced
}

#use height cutoff to remove
cutTreeStatic = function(hiercluster,heightcutoff=0.99, minsize1=50) {

    # here we define modules by using a height cut-off for the branches
    labelpred= cutree(hiercluster,h=heightcutoff)
    sort1=-sort(-table(labelpred))
    sort1
    modulename= as.numeric(names(sort1))
    modulebranch= sort1 > minsize1
    no.modules=sum(modulebranch)

    colorhelp = rep(-1, length(labelpred) )
    if ( no.modules==0){
        print("No module detected\n")
    }
    else{
        for (i in c(1:no.modules)) {
            colorhelp=ifelse(labelpred==modulename[i],i ,colorhelp)
        }
    }
    colorhelp
}

#leftOrright >0 : running length (with same sign) to right, otherwise to the left
#mysign = -1: negative value, mysign = 1: positive value
runlengthSign = function(mysequence, leftOrright=-1, mysign=-1){
    seqlen = length(mysequence)
    if(leftOrright<0){
        pseq = rev(mysequence)
    }else{
        pseq = mysequence
    }

    if(mysign<0){ #see where the first POSITIVE number occurs
        nonezero.bool = (pseq > 0)
    }else{ #see where the first NEGATIVE number occur
        nonezero.bool = (pseq < 0)
    }
    if( sum(nonezero.bool) > 0){
        runlength = min( c(1:seqlen)[nonezero.bool] ) - 1
    }else{
        runlength = 0
    }
}
}

```

```

#"0" is for grey module
assignModuleColor = function(labelpred, minsize1=50, anameallmodules=FALSE,
auseblackwhite=FALSE) {
  # here we define modules by using a height cut-off for the branches
  #labelpred= cutree(hiercluster,h=heightcutoff)
  #cat(labelpred)

  #"0", grey module doesn't participate color assignment, directly assigned as "grey"
  labelpredNoZero = labelpred[ labelpred >0 ]
  sort1=-sort(-table(labelpredNoZero))
  sort1
  modulename= as.numeric(names(sort1))
  modulebranch= sort1 > minsize1
  no.modules=sum(modulebranch)

  # now we assume that there are fewer than 10 modules
  nongrayvec=c(c(1:151),c(254:259),c(362:657))
  colorcode=as.character(sample(colors()[nongrayvec],no.modules))

  #("turquoise","blue","brown","yellow","green","red","black","pink","magenta","purple","gr
eenyellow","tan","salmon","cyan", #"midnightblue", "lightcyan","grey60", "lightgreen",
#"lightyellow","coral","wheat","seagreen","rosybrown","slateblue","darkolivegreen","maroo
n","peachpuff","olivedrab","honeydew","aquamarine","azure","bisque","burlywood","cadetblu
e","chartreuse","darkgoldenrod","darkkhaki","darkorchid1","deeppink","firebrick","forestg
reen","indianred","lawngreen","mintcream","mistyrose","orange","orangered1","palegreen","
sienna4","slategray1","thistle","tomato4","violetred4","plum4","palevioletred","orange4",
"mediumspringgreen","lemonchiffon","moccasin","ivory","darkseagreen4","gainsboro","brown4
")

  #"grey" means not in any module;
  colorhelp=rep("grey",length(labelpred))
  if ( no.modules==0){
    print("No mudule detected\n")
  }
  else{
    if ( no.modules > length(colorcode) ){
      print( paste("Too many modules \n", as.character(no.modules)) )
    }

    if ( (anameallmodules==FALSE) || (no.modules <=length(colorcode)) ){
      labeledModules = min(no.modules, length(colorcode) )
      for (i in c(1:labeledModules)) {
        colorhelp=ifelse(labelpred==modulename[i],colorcode[i],colorhelp)
      }
      colorhelp=factor(colorhelp,levels=c(colorcode[1:labeledModules],"grey"))
    }else{#nameallmodules==TRUE and no.modules >length(colorcode)
      maxcolors=length(colorcode)
      labeledModules = no.modules
      extracolors=NULL
      blackwhite=c("red", "black")
      for(i in c((maxcolors+1):no.modules)){
        if(auseblackwhite==FALSE){
          icolor=paste("module", as.character(i), sep="")
        }else{#use balck white alternatively represent extra colors, for display
only
          #here we use the ordered label to avoid put the same color for two
neighboring clusters
          icolor=blackwhite[1+(as.integer(modulename[i])%2) ]
        }
      }
    }
  }
}

```

```

        extracolors=c(extracolors, icolor)
    }

    #combine the true-color code and the extra colorcode into a uniform colorcode
for
    #color assignment
    allcolorcode=c(colorcode, extracolors)

    for (i in c(1:labeledModules)) {
        colorhelp=ifelse(labelpred==modulename[i],allcolorcode[i],colorhelp)
    }
    colorhelp=factor(colorhelp,levels=c(allcolorcode[1:labeledModules],"grey"))
}

}

colorhelp
}

#locate the start/end positions of each cluster in the ordered cluster label sequence
#where "-1" indicating no cluster
#3-1 -1 1 1 1 1 2 2 2
#3 3 -1-1 1 1 1 1 2 2 2 (shift)
#-----
#0-4  0 2 0 0 0 1 0 0 0 (difference)
#      *      * @
locateCluster = function(clusterlabels)
{
    no.nodes = length(clusterlabels)
    clusterlabels.shift = c(clusterlabels[1], c(clusterlabels[1:(no.nodes-1)]) )

    #a non-zero point is the start point of a cluster and it previous point is the end point
of the previous cluster
    label.diff = abs(clusterlabels - clusterlabels.shift)

    #process the first and last positions as start/end points if they belong to a cluster
instead of no cluster "-1"
    if(clusterlabels[1] >0) {label.diff[1]=1}
    if(clusterlabels[no.nodes]>0) {label.diff[no.nodes]=1}

    flagpoints.bool = label.diff > 0
    flagpoints = c(1:no.nodes)[flagpoints.bool]
    no.points = length(flagpoints)

    myclupos=NULL
    for(i in c(1:(no.points-1)) ){
        idx = flagpoints[i]
        if(clusterlabels[idx]>0){
            myclupos = rbind(myclupos, c(idx, flagpoints[i+1]-1) )
        }
    }
    myclupos
}

#input is the cluster demdrogram of an individual cluster, we want to find its embedded
subclusters
#execution order: mean-height ==> (mean+max)/2 ==> (mean+min)/2
#useMean: =0 ~ use mean-height as calibration line

```

```

#           =1 ~ use (mean+max)/2 as calibration line to detect relatively a small cluster
sitting on the head of a bigger one,
#           so mean-height is too low to detect the two modules.
#           =-1~ use (mean+min)/2 as calibration line to detect relatively a small cluster
sitting on the tail of a bigger one,
#           so mean-height & (mean+max)/2 are too high to detect the two
modules

```

```

processInvididualCluster = function(clusterDemdroHei, cminModuleSize=50,
cminAttachModuleSize=100, minTailRunlength=12, useMean=0){
  #for debug: use all genes
  #clusterDemdroHei =demdroHei.order

  no.cnodes = dim(clusterDemdroHei)[1]

  cmaxhei = max(clusterDemdroHei[, 2])
  cminhei = min(clusterDemdroHei[, 2])

  cmeanhei = mean(clusterDemdroHei[, 2])
  cmidhei = (cmeanhei + cmaxhei)/2.0
  cdwnhei = (cmeanhei + cminhei)/2.0

  if (useMean==1){
    comphei = cmidhei
  }else if (useMean==-1){
    comphei = cdwnhei
  }else{ #normal case
    comphei = cmeanhei
  }

  # compute height difference with mean height
  heidiff = clusterDemdroHei[,2] - comphei
  heidiff.shift = shiftSequence(heidiff, -1)

  # get cut positions
  # detect the end point of a cluster, whose height should be less than meanhei
  # and the node behind it is the start point of the next cluster which has a height
above meanhei
  cuts.bool = (heidiff<0) & (heidiff.shift > 0)
  cuts.bool[1] = TRUE
  cuts.bool[no.cnodes] = TRUE

  if(sum(cuts.bool)==2){
    if (useMean==0){
      new.clupos=processInvididualCluster(clusterDemdroHei=clusterDemdroHei,
cminModuleSize=cminModuleSize,
cminAttachModuleSize=cminAttachModuleSize,
useMean=1)
    }else if(useMean==1){
      new.clupos=processInvididualCluster(clusterDemdroHei=clusterDemdroHei,
cminModuleSize=cminModuleSize,
cminAttachModuleSize=cminAttachModuleSize,
useMean=-1)
    }else{
      new.clupos = rbind(c(1, no.cnodes))
    }
  }
  return (new.clupos)
}

#a good candidate cluster-end point should have significant # of ahead nodes with
head < meanHei

```

```

cutindex =c(1:no.cnodes)[cuts.bool]
no.cutps = length(cutindex)
runlens = rep(999, no.cutps)
cuts.bool2 = cuts.bool
for(i in c(2:(no.cutps-1)) ){
  seq = c( (cutindex[i-1]+1):cutindex[i] )
  runlens[i] = runlengthSign(heidiff[seq], leftOrright=-1, mysign=-1)

  if(runlens[i] < minTailRunlength){
    #cat("run length=", runlens[i], "\n")
    cuts.bool2[ cutindex[i] ] = FALSE
  }
}

#attach SMALL cluster to the left-side BIG cluster if the small one has smaller mean
height
cuts.bool3=cuts.bool2
if(sum(cuts.bool2) > 3) {
  curj = 2
  while (l==1){
    cutindex2 =c(1:no.cnodes)[cuts.bool2]
    no.clus = length(cutindex2) -1
    if (curj>no.clus){
      break
    }
    pre.sdx = cutindex2[ curj-1 ]+1 #previous module start point
    pre.edx = cutindex2[ curj ] #previous module end point
    pre.module.size = pre.edx - pre.sdx +1
    pre.module.hei = mean(clusterDemdoroHei[c(pre.sdx:pre.edx) , 2])

    cur.sdx = cutindex2[ curj ]+1 #previous module start point
    cur.edx = cutindex2[ curj+1 ] #previous module end point
    cur.module.size = cur.edx - cur.sdx +1
    cur.module.hei = mean(clusterDemdoroHei[c(cur.sdx:cur.edx) , 2])

    #merge to the leftside major module, don't change the current index "curj"
    #if( (pre.module.size
>minAttachModuleSize)&(cur.module.hei<pre.module.hei)&(cur.module.size<minAttachModuleSize) )){
      if( (cur.module.hei<pre.module.hei)&(cur.module.size<cminAttachModuleSize) ){
        cuts.bool2[ cutindex2[curj] ] = FALSE
      }else{ #consider next cluster
        curj = curj + 1
      }
    }#while
  }#if

  cutindex2 =c(1:no.cnodes)[cuts.bool2]
  no.cutps = length(cutindex2)

  #we don't want to lose the small cluster at the tail, attach it to the previous big
cluster
#cat("Lclu= ", cutindex2[no.cutps]-cutindex2[no.cutps-1]+1, "\n")
if(no.cutps > 2){
  if( (cutindex2[no.cutps] - cutindex2[no.cutps-1]+1) < cminModuleSize ){
    cuts.bool2[ cutindex2[no.cutps-1] ] =FALSE
  }
}

if(l==2){
  myseqnce = c(2300:3000)

```

```

cutdisp = ifelse(cuts.bool2==T, "red","grey" )
#re-order to the normal one with sequential singleton index
par(mfrow=c(3,1), mar=c(0,0,0,0) )
plot(hlrow, labels=F, xlab="",ylab="",main="",sub="",axes = F)
barplot(heidiff[myseqnce],
        col= "black", space=0,
        border=F,main="", axes = F, axisnames = F)
barplot(height=rep(1, length(cutdisp[myseqnce])),
        col= as.character(cutdisp[myseqnce]), space=0,
        border=F,main="", axes = F, axisnames = F)
par(mfrow=c(1,1), mar=c(5, 4, 4, 2) + 0.1)
}

cutindex2 = c(1:no.cnodes)[cuts.bool2]
cutindex2[1]=cutindex2[1]-1 #the first
no.cutps2 = length(cutindex2)

if(no.cutps2 > 2){
  new.clupos = cbind( cutindex2[c(1:(no.cutps2-1))]+1, cutindex2[c(2:no.cutps2)] )
}else{
  new.clupos = cbind( 1, no.cnodes)
}

if ( dim(new.clupos)[1] == 1 ){
  if (useMean==0){
    new.clupos=processInvididualCluster(clusterDemdroHei=clusterDemdroHei,
cminModuleSize=cminModuleSize,
                                     cminAttachModuleSize=cminAttachModuleSize,
                                     useMean=1)
  }else if(useMean==1){
    new.clupos=processInvididualCluster(clusterDemdroHei=clusterDemdroHei,
cminModuleSize=cminModuleSize,
                                     cminAttachModuleSize=cminAttachModuleSize,
                                     useMean=-1)
  }
}
new.clupos
}

```

```

findClustersSignificant=function(mysequence, modulecolor)
{
  modnames= names( table(modulecolor) )
  mysize = length(modulecolor)
  validseq = rep(TRUE, mysize)
  for (each in modnames ){
    mybool = (modulecolor==each)
    mymodulesig = mysequence[mybool]
    mydiff = abs(mymodulesig - mymodulesig[1])
    if(sum(mydiff)==0){
      validseq = ifelse(mybool==TRUE, FALSE, validseq)
    }
  }
  validseq
}

```

```

#leftOrright >0 : running length (with same sign) to right, otherwise to the left
#mysign = -1: negative value, mysign = 1: positive value
runlengthSign = function(mysequence, leftOrright=-1, mysign=-1){
  seqlen = length(mysequence)
  if(leftOrright<0){
    pseq = rev(mysequence)
  }else{
    pseq = mysequence
  }

  if(mysign<0){ #see where the first POSITIVE number occurs
    nonezero.bool = (pseq > 0)
  }else{ #see where the first NEGATIVE number occur
    nonezero.bool = (pseq < 0)
  }
  if( sum(nonezero.bool) > 0){
    runlength = min( c(1:seqlen)[nonezero.bool] ) - 1
  }else{
    runlength = 0
  }
}

```

```

#delta >0 : shift to right, otherwise to the left
shiftSequence = function(mysequence, delta){
  seqlen = length(mysequence)
  if(delta>0){
    finalseq=c(mysequence[1:delta], mysequence[1:(seqlen-delta)])
  }else{
    posdelta = -delta
    finalseq=c(mysequence[(posdelta+1):seqlen], mysequence[(seqlen-
posdelta+1):seqlen])
  }
  finalseq
}

```

```

#no of neighbors behind and before the point used for average
averageSequence=function(mysequence, noneighbors){
  sumseq = mysequence
  for(i in c(1:noneighbors)){
    iseq = shiftSequence(mysequence, i)
    sumseq = sumseq + iseq
    iseq = shiftSequence(mysequence, -i)
    sumseq = sumseq + iseq
  }
  sumseq = sumseq/(1+2*noneighbors)
}

```

```

#delta >0 : shift to right, otherwise to the left
shiftSequence = function(mysequence, delta){
  seqlen = length(mysequence)
  if(delta>0){
    finalseq=c(mysequence[1:delta], mysequence[1:(seqlen-delta)])
  }else{
    posdelta = -delta
    finalseq=c(mysequence[(posdelta+1):seqlen], mysequence[(seqlen-
posdelta+1):seqlen])
  }
  finalseq
}

```

```

}

#find the middle of each cluster and label the middle position with the corresponding
color
getDisplayColorSequence=function(colordered){
  mylen = length(colordered)
  colordered2 = c(colordered[1], colordered[1:(mylen-1)] )
  colordiff = (colordered != colordered2)
  colordiff[1] = TRUE
  colordiff[mylen] = TRUE
  #mydispcolor = ifelse(colordiff==TRUE, colordered, "")
  mydispcolor = rep("", mylen)
  mytrueseq = c(1:mylen)[colordiff]
  for (i in c(1:(length(mytrueseq)-1)) ){
    midi = (mytrueseq[i] + mytrueseq[i+1])/2
    mydispcolor[midi] = colordered[midi]
  }
  fdispcolor = ifelse(mydispcolor=="grey", "", mydispcolor)
  fdispcolor
}

#use height cutoff to remove
cutTreeStatic = function(hiercluster,heightcutoff=0.99, minsize1=50) {

  # here we define modules by using a height cut-off for the branches
  labelpred= cutree(hiercluster,h=heightcutoff)
  sort1=-sort(-table(labelpred))
  sort1
  modulename= as.numeric(names(sort1))
  modulebranch= sort1 > minsize1
  no.modules=sum(modulebranch)

  colorhelp = rep(-1, length(labelpred) )
  if ( no.modules==0){
    print("No module detected\n")
  }
  else{
    for (i in c(1:no.modules)) {
      colorhelp=ifelse(labelpred==modulename[i],i ,colorhelp)
    }
  }
  colorhelp
}

#merge the minor cluster into the major cluster
merge2Clusters = function(mycolorcode, mainclusterColor, minorclusterColor){
  mycolorcode2 = ifelse(as.character(mycolorcode)==minorclusterColor, mainclusterColor,
as.character(mycolorcode) )
  fcolorcode =factor(mycolorcode2)
  fcolorcode
}

```

```

#####
#####
# I) GENERAL STATISTICAL FUNCTIONS

```

```

mean1=function(x) mean(x,na.rm=T)
var1=function(x) var(x,na.rm=T)

```

```

if (exists("scatterplot1") ) rm(scatterplot1);
scatterplot1=function(x,y, title1="",coll="black",xlab1="x",ylab1="y" , cex1=1,
cex.axis1=1.5,cex.lab1=1.5, cex.main1=1.5 ,ylim1=-1 ){
cor1=signif(cor(x,y,use="p",method="s"),2)
corp=signif(cor.test(x,y,use="p",method="s")$p.value,2)
if (corp<10^(-20) ) corp="<10^{-20}"
if ( length(ylim1[1])==2) {plot(x,y, main=paste(title1,"cor=",
cor1,"p=",corp),col=coll,xlab=xlab1,ylab=ylab1, cex=cex1,
cex.axis=cex.axis1,cex.lab=cex.lab1, cex.main=cex.main1,ylim=ylim1)} else {
plot(x,y, main=paste(title1,"cor=",
cor1,"p=",corp),col=as.character(coll),xlab=xlab1,ylab=ylab1, cex=cex1,
cex.axis=cex.axis1,cex.lab=cex.lab1, cex.main=cex.main1)}
}

```

```

## ModuleQCnoGrey function: PC1, minTO, maxCOR, density, density index, and density p-
value (permutation test).

```

```

if(exists("ModuleQCnoGrey")) rm(ModuleQCnoGrey);
ModuleQCnoGrey=function(couleur,pcs1,distTOM1,datExpr1,hierTOM1,perms){
moduledf=data.frame(table(couleur))
ordervec=order(as.character(moduledf[,1]))
datpc=data.frame(moduledf[ordervec,],t(pcs1$varexplained[1,]))
datminTO=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
whichcolor=as.character(moduledf[ordervec,][i,1])
datminTO[i,1]=min(distTOM1[ couleur==whichcolor,couleur==whichcolor])
}
collect_garbage()
Pearson=cor(datExpr1,use="p")
diag(Pearson)=0
collect_garbage()
datmaxCOR=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
whichcolor=as.character(moduledf[ordervec,][i,1])
datmaxCOR[i,1]=max(abs(Pearson[ couleur==whichcolor,couleur==whichcolor]))
}
rm(Pearson)
collect_garbage()
simTOM=1-distTOM1
datDENSITY=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
whichcolor=as.character(moduledf[ordervec,][i,1])
nogenes=length(couleur[ couleur==whichcolor])
datDENSITY[i,1]=sum(simTOM[ couleur==whichcolor,couleur==whichcolor])/(nogenes*(nogenes-
1))
}
datpcDENSITY=data.frame(datpc,datminTO,datmaxCOR,datDENSITY)
greyrow=is.element(as.character(datpcDENSITY[,1]),"grey")
densityindex=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
densityindex[i,1]=datpcDENSITY[i,6]/datpcDENSITY[greyrow,6]
}
datpcDENSITY=data.frame(datpcDENSITY,densityindex)
moduledf2=data.frame(unique(couleur[hierTOM1$order]))

```

```

rankcolors=rank(as.character(moduledf2[,1]))
collect_garbage()
counter=c(seq(1,length(unique(couleur)),by=1))
permDENSITY=matrix(nrow=length(unique(couleur)),ncol=perms+1)
for (j in c(1:length(unique(couleur)))){
  whichcolor=as.character(moduledf[ordervec,][j,1])
  if (whichcolor=="grey") {
    for (i in c(2:(perms+1))){
      permDENSITY[j,1]=whichcolor
      permDENSITY[j,i]="NA"
    }
  }else{for (i in c(2:(perms+1))){
    nogenes=length(couleur[couleur==whichcolor])
    randomsample=sample(c(1:length(datExpr1[1,])),nogenes,replace=F)
    permDENSITY[j,1]=whichcolor

permDENSITY[j,i]=sum(simTOM[randomsample,randomsample])/(nogenes*(nogenes-1))
  }
  if (length(counter[counter==j])>0) {print(paste(whichcolor,j)); timestamp()}
}
collect_garbage()
dpvalues=matrix(nrow=length(unique(couleur)),ncol=2)
for (i in c(1:length(unique(couleur)))){
  whichcolor=as.character(moduledf[ordervec,][i,1])
  dpvalues[i,1]=whichcolor

dpvalues[i,2]=length(permDENSITY[i,2:perms+1][permDENSITY[i,2:perms+1]>datpcDENSITY[i,6]]
)/perms
}
datpcDENSITY=data.frame(cbind(datpcDENSITY[rankcolors,],I(dpvalues[,2][rankcolors]),c(1:l
ength(unique(couleur)))))
colnames(datpcDENSITY)=c("Module","Freq","PC1_Var","Min_TO","Max_COR","Density","Density_
index","P.value","Tree_order")
rm(simTOM)
collect_garbage()
datpcDENSITY
}

```

ModuleQCnoPerm function: PC1, minTO, maxCOR, density, and density index.

```

if(exists("ModuleQCnoPerm")) rm(ModuleQCnoPerm);
ModuleQCnoPerm=function(couleur,pcs1,distTOM1,datExpr1,hierTOM1){
moduledf=data.frame(table(couleur))
ordervec=order(as.character(moduledf[,1]))
datpc=data.frame(moduledf[ordervec,],t(pcs1$varexplained[1,]))
datminTO=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
  whichcolor=as.character(moduledf[ordervec,][i,1])
  datminTO[i,1]=min(distTOM1[couleur==whichcolor,couleur==whichcolor])
}
collect_garbage()
Pearson=cor(datExpr1,use="p")
diag(Pearson)=0
collect_garbage()
datmaxCOR=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
  whichcolor=as.character(moduledf[ordervec,][i,1])
  datmaxCOR[i,1]=max(abs(Pearson[couleur==whichcolor,couleur==whichcolor]))
}

```

```

}
rm(Pearson)
collect_garbage()
simTOM=1-distTOM1
datDENSITY=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
  whichcolor=as.character(moduledf[ordervec,][i,1])
  nogenes=length(couleur[couleur==whichcolor])
  datDENSITY[i,1]=sum(simTOM[couleur==whichcolor,couleur==whichcolor])/(nogenes*(nogenes-1))
}
datpcDENSITY=data.frame(datpc,datminTO,datmaxCOR,datDENSITY)
greyrow=is.element(as.character(datpcDENSITY[,1]),"grey")
densityindex=matrix(nrow=length(unique(couleur)),ncol=1)
for (i in c(1:length(unique(couleur)))){
  densityindex[i,1]=datpcDENSITY[i,6]/datpcDENSITY[greyrow,6]
}
datpcDENSITY=data.frame(datpcDENSITY,densityindex)
moduledf2=data.frame(unique(couleur[hierTOM1$order]))
rankcolors=rank(as.character(moduledf2[,1]))
collect_garbage()
datpcDENSITY=data.frame(cbind(datpcDENSITY[rankcolors,],c(1:length(unique(couleur)))))
colnames(datpcDENSITY)=c("Module","Freq","PC1_Var","Min_TO","Max_COR","Density","Density_index","Tree_order")
rm(simTOM)
collect_garbage()
datpcDENSITY
}

```

#The function DegreeInOutMO computes for each gene

#a) the total number of connections,

#b) the number of connections with genes within its module,

#c) the signed correlation of the gene with its eigengene

if (exists("DegreeInOutMO")) rm(DegreeInOutMO);

DegreeInOutMO=function(datExpr1, adj1, couleur, pcs1) {

no.nodes=length(couleur)

couleurlevels=levels(factor(couleur))

couleurlevels=couleurlevels[order(as.character(couleurlevels))]

no.levels=length(couleurlevels)

kWithin=rep(-666,no.nodes)

eigencorr=rep(-666,no.nodes)

eigenpval=rep(-666,no.nodes)

diag(adj1)=0

for (i in c(1:no.levels)) {

rest1=couleur==couleurlevels[i];

kWithin[rest1]=apply(adj1[rest1,rest1],2,sum)

datExprmod=datExpr1[,rest1]

for (j in c(1:length(rest1[rest1]))) {

 eigencorr[rest1][j]=cor.test(datExprmod[,j],pcs1\$PrinComps[,i],method="p")\$estimate

 eigenpval[rest1][j]=cor.test(datExprmod[,j],pcs1\$PrinComps[,i],method="p")\$p.value

}

}

kTotal= apply(adj1,2,sum)

data.frame(kTotal,kWithin,eigencorr,eigenpval)

}