```
##WGCNA
#http://amphipod.hatenablog.com/entry/2018/01/05/155031
#Data input and cleaning

#=================================================================
=======================
#
#  Code chunk 1
#
#=================================================================
=======================

# Load the WGCNA package
library(WGCNA);
datExpr0 <- read.csv("path/to/dataset.csv")
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
# Take a quick look at what is in the data set:
dim(datExpr0);
names(datExpr0);
head(datExpr0)
#=================================================================
=======================
#
#  Code chunk 2
#
#=================================================================
=======================

cell <- factor(rep(c("Bm1", "naive", "preGCB", "memory"), 4))
status <- factor(c(rep("HC", 24), rep("pSS", 24)))
ID <- factor(1:48)
new_colname                                                    <-
c("Bm1.HC.1","naive.HC.1","preGCB.HC.1","memory.HC.1",
              "Bm1.HC.2",
"naive.HC.2","preGCB.HC.2","memory.HC.2",
              "Bm1.HC.3",
"naive.HC.3","preGCB.HC.3","memory.HC.3",

"Bm1.HC.4","naive.HC.4","preGCB.HC.4","memory.HC.4",

"Bm1.HC.5","naive.HC.5","preGCB.HC.5","memory.HC.5",
              "Bm1.HC.6","naive.HC.6","preGCB.HC.6",
"memory.HC.6",

"Bm1.SjS.1","naive.SjS.1","preGCB.SjS.1","memory.SjS.1",

"Bm1.SjS.2","naive.SjS.2","preGCB.SjS.2","memory.SjS.2",

"Bm1.SjS.3","naive.SjS.3","preGCB.SjS.3","memory.SjS.3",

"Bm1.SjS.4","naive.SjS.4","preGCB.SjS.4","memory.SjS.4",

"Bm1.SjS.5","naive.SjS.5","preGCB.SjS.5","memory.SjS.5",

"Bm1.SjS.6","naive.SjS.6","preGCB.SjS.6","memory.SjS.6"
)
rownames(datExpr0) = new_colname
head(datExpr0)
names(datExpr0)
str(datExpr0)
datExpr0.SjS <- datExpr0[25:48, ]
datExpr0.HC <- datExpr0[1:24, ]
#===============================================================
=======================
#
#  Code chunk 3
#
#===============================================================
=======================
gsg = goodSamplesGenes(datExpr0, verbose = 3);
gsg$allOK


#=============================================================
======================
#
#  Code chunk 4
#
#=============================================================
======================


if (!gsg$allOK)
{
  # Optionally, print the gene and sample names that were
removed:
  if (sum(!gsg$goodGenes)>0)
    printFlush(paste("Removing                              genes:",
paste(names(datExpr0)[!gsg$goodGenes], collapse = ", ")));
  if (sum(!gsg$goodSamples)>0)
    printFlush(paste("Removing                             samples:",
paste(rownames(datExpr0)[!gsg$goodSamples], collapse = ", ")));
  # Remove the offending genes and samples from the data:
  datExpr0 = datExpr0[gsg$goodSamples, gsg$goodGenes]
}


#=============================================================
======================
#
#  Code chunk 5
#
#=============================================================
=======================


sampleTree = hclust(dist(datExpr0), method = "average");
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree, main = "Sample clustering to detect outliers",
sub="", xlab="", cex.lab = 1.5,
    cex.axis = 1.5, cex.main = 2)


#SjS
sampleTree.SjS  =  hclust(dist(datExpr0[25:48,  ]),  method  =
"average");
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree.SjS,  main  =  "Sample  clustering  to  detect
outliers", sub="", xlab="", cex.lab = 1.5,
    cex.axis = 1.5, cex.main = 2)

#HC
sampleTree.HC=   hclust(dist(datExpr0[1:24,   ]),   method   =
"average");
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree.HC,  main  =  "Sample  clustering  to  detect
outliers", sub="", xlab="", cex.lab = 1.5,
    cex.axis = 1.5, cex.main = 2)

#==============================================================
=======================
#
#  Code chunk 6
#
#==============================================================
=======================
# Plot a line to show the cut
abline(h = 173, col = "red");
# Determine cluster under the line
clust = cutreeStatic(sampleTree, cutHeight = 173, minSize = 10)
table(clust)
# clust 1 contains the samples we want to keep.
keepSamples = (clust==1)
datExpr = datExpr0[keepSamples, ]
nGenes = ncol(datExpr)
nSamples = nrow(datExpr)


##SjS
# Determine cluster under the line
clust.SjS  =  cutreeStatic(sampleTree.SjS,  cutHeight  =  173,
minSize = 10)
table(clust.SjS)
# clust 1 contains the samples we want to keep.
keepSamples.SjS = (clust.SjS==1)
datExpr.SjS = datExpr0.SjS[keepSamples.SjS, ]
nGenes.SjS = ncol(datExpr.SjS)
nSamples.SjS = nrow(datExpr.SjS)

##HC
# Determine cluster under the line
clust.HC = cutreeStatic(sampleTree.HC, cutHeight = 173, minSize
= 10)
table(clust.HC)
# clust 1 contains the samples we want to keep.
keepSamples.HC = (clust.HC==1)
datExpr.HC = datExpr0.HC[keepSamples.HC, ]
nGenes.HC = ncol(datExpr.HC)
nSamples.HC = nrow(datExpr.HC)
```

```
#=================================================================
=======================
#
#  Code chunk 7
#
#=================================================================
=======================


traitData                                                       =
read.csv("path/to/Clinical_information_dataset.csv");
dim(traitData)
names(traitData)

# remove columns that hold information we do not need.
Clinical_Variable         <-         c(         "sample","age",
"disease_duration","Ocular",
"Raynaud","SSB_positive","ESSDAI","Hematological" ,"Biological",
"ANA"  ,"IgG",  "IgA"  ,  "IgM","CRP"  ,"ESR"  ,  "AMY",  "Lym",
"SS.A"                 ,"SS.B","RF","C3"                ,"C4",
"CH50","PG_uptake","SMG_uptake","PG_excretion" ,"SMG_excretion"
)
allTraits = traitData[, Clinical_Variable]
dim(allTraits)
names(allTraits)
head(allTraits)


# Form a data frame analogous to expression data that will hold
the clinical traits.

femaleSamples = rownames(datExpr0.SjS);
traitRows = match(femaleSamples, allTraits$X...sample)
datTraits = allTraits[traitRows, -1];
rownames(datTraits) = allTraits[traitRows, 1];
head(datTraits)
head(allTraits$X...samp)
head(femaleSamples)
collectGarbage();


#=================================================================
=======================
#
#  Code chunk 8
#
#=================================================================
=======================


# Re-cluster samples
sampleTree2 = hclust(dist(datExpr0.SjS), method = "average")
# Convert traits to a color representation: white means low, red
means high, grey means missing entry
traitColors = numbers2colors(datTraits, signed = FALSE);
# Plot the sample dendrogram and the colors underneath.
plotDendroAndColors(sampleTree2, traitColors,
                 groupLabels = names(datTraits),
                 main = "Sample dendrogram and trait heatmap")


#=================================================================
=======================
#
#  Code chunk 9
#
#=================================================================
=======================


save(datExpr.SjS,          datTraits,          file          =
"AfterFilterByExpressAndCV_KIRARI.RData")
```

```
##Automatic network construction and module detection


#==============================================================
=======================
#
#  Code chunk 2
#
#==============================================================
=======================


# Choose a set of soft-thresholding powers
powers = c(c(1:10), seq(from = 12, to=20, by=2))
# Call the network topology analysis function
sft.SjS = pickSoftThreshold(datExpr.SjS, powerVector = powers,
verbose = 5)
# Plot the results:
sizeGrWindow(9, 5)
par(mfrow = c(1,2));
cex1 = 0.9;
# Scale-free topology fit index as a function of the soft-
thresholding power
plot(sft.SjS$fitIndices[,1],                                -
sign(sft.SjS$fitIndices[,3])*sft.SjS$fitIndices[,2],
    xlab="Soft Threshold (power)",ylab="Scale Free Topology
Model Fit,signed R^2",type="n",
    main = paste("Scale independence"));
text(sft.SjS$fitIndices[,1],                                -
sign(sft.SjS$fitIndices[,3])*sft.SjS$fitIndices[,2],
    labels=powers,cex=cex1,col="red");
# this line corresponds to using an R^2 cut-off of h
```

```
abline(h=0.90,col="red")
# Mean connectivity as a function of the soft-thresholding power
plot(sft.SjS$fitIndices[,1], sft.SjS$fitIndices[,5],
    xlab="Soft  Threshold  (power)",ylab="Mean  Connectivity",
type="n",
    main = paste("Mean connectivity"))
text(sft.SjS$fitIndices[,1],           sft.SjS$fitIndices[,5],
labels=powers, cex=cex1,col="red")


##HC
# Call the network topology analysis function
sft.HC  =  pickSoftThreshold(datExpr.HC,  powerVector  =  powers,
verbose = 5)
# Plot the results:
sizeGrWindow(9, 5)
par(mfrow = c(1,2));
cex1 = 0.9;
#  Scale-free  topology  fit  index  as  a  function  of  the  soft-
thresholding power
plot(sft.HC$fitIndices[,1],                                     -
sign(sft.HC$fitIndices[,3])*sft.HC$fitIndices[,2],
    xlab="Soft  Threshold  (power)",ylab="Scale  Free  Topology
Model Fit,signed R^2",type="n",
    main = paste("Scale independence"));
text(sft.HC$fitIndices[,1],                                     -
sign(sft.HC$fitIndices[,3])*sft.HC$fitIndices[,2],
    labels=powers,cex=cex1,col="red");
# this line corresponds to using an R^2 cut-off of h
abline(h=0.90,col="red")
# Mean connectivity as a function of the soft-thresholding power
plot(sft.HC$fitIndices[,1], sft.HC$fitIndices[,5],
    xlab="Soft  Threshold  (power)",ylab="Mean  Connectivity",
type="n",
    main = paste("Mean connectivity"))
text(sft.HC$fitIndices[,1],             sft.HC$fitIndices[,5],
labels=powers, cex=cex1,col="red")


#=============================================================
======================
#
#  Code chunk 3
#
#=============================================================
======================

#SjS
net.SjS = blockwiseModules(datExpr.SjS, power = 10,
                      TOMType = "unsigned", minModuleSize = 30,
                      reassignThreshold  =  0,  mergeCutHeight  =
0.25,
                      numericLabels = TRUE, pamRespectsDendro =
FALSE,
                      saveTOMs = TRUE,
                      saveTOMFileBase = "KIRARI.SjS",
                      verbose = 3)

#HC
net.HC = blockwiseModules(datExpr.HC, power = 10,
                     TOMType = "unsigned", minModuleSize = 30,
                     reassignThreshold  =  0,  mergeCutHeight  =
0.25,
                     numericLabels = TRUE, pamRespectsDendro =
FALSE,
                     saveTOMs = TRUE,
                     saveTOMFileBase = "KIRARI.HC",
                     verbose = 3)

#=============================================================
======================
#
```

```
#  Code chunk 4
#
#==============================================================
=======================

##SjS
# open a graphics window
sizeGrWindow(12, 9)
# Convert labels to colors for plotting
mergedColors.SjS = labels2colors(net.SjS$colors)
# Plot the dendrogram and the module colors underneath
plotDendroAndColors(net.SjS$dendrograms[[1]],
mergedColors.SjS[net.SjS$blockGenes[[1]]],
                 "Module colors",
                 dendroLabels = FALSE, hang = 0.03,
                 addGuide = TRUE, guideHang = 0.05)

#HC
# open a graphics window
sizeGrWindow(12, 9)
# Convert labels to colors for plotting
mergedColors.HC = labels2colors(net.HC$colors)
# Plot the dendrogram and the module colors underneath
plotDendroAndColors(net.HC$dendrograms[[1]],
mergedColors.HC[net.HC$blockGenes[[1]]],
                 "Module colors",
                 dendroLabels = FALSE, hang = 0.03,
                 addGuide = TRUE, guideHang = 0.05)


#==============================================================
=======================
#
#  Code chunk 5
#
#==============================================================
=======================

#SjS
moduleLabels.SjS = net.SjS$colors
moduleColors.SjS = labels2colors(net.SjS$colors)
MEs.SjS = net.SjS$MEs;
geneTree.SjS = net.SjS$dendrograms[[1]];
save(MEs.SjS, moduleLabels.SjS, moduleColors.SjS, geneTree.SjS,
    file        =        "AfterFilterByExpressAndCV_KIRARI.SjS-
networkConstruction-auto.RData")


#HC
moduleLabels.HC = net.HC$colors
moduleColors.HC = labels2colors(net.HC$colors)
MEs.HC = net.HC$MEs;
geneTree.HC = net.HC$dendrograms[[1]];
save(MEs.HC, moduleLabels.HC, moduleColors.HC, geneTree.HC,
    file        =        "AfterFilterByExpressAndCV_KIRARI.HC-
networkConstruction-auto.RData")




#Relating  modules  to  external  information  and  identifying
important

##SjS
# Define numbers of genes and samples
nGenes.SjS = ncol(datExpr.SjS);
nSamples.SjS = nrow(datExpr.SjS);
# Recalculate MEs with color labels
MEs0.SjS             =             moduleEigengenes(datExpr.SjS,
moduleColors.SjS)$eigengenes
MEs.SjS = orderMEs(MEs0.SjS)
moduleTraitCor.SjS = cor(MEs.SjS, datTraits, use = "p");
```

```
moduleTraitPvalue.SjS   =   corPvalueStudent(moduleTraitCor.SjS,
nSamples.SjS);


cell <- factor(rep(c("Bm1", "naive", "preGCB", "memory"), 6))
ID <- factor(c(rep(1, 4),rep(2, 4), rep(3, 4), rep(4, 4),rep(5,
4),rep(6, 4)))
MEG.SjS <- cbind(MEs0.SjS, cell, ID)



##HC
# Define numbers of genes and samples
nGenes.HC = ncol(datExpr.HC);
nSamples.HC = nrow(datExpr.HC);
# Recalculate MEs with color labels
MEs0.HC              =              moduleEigengenes(datExpr.HC,
moduleColors.HC)$eigengenes
MEs.HC = orderMEs(MEs0.HC)
moduleTraitCor.HC = cor(MEs.HC, datTraits, use = "p");
moduleTraitPvalue.HC    =    corPvalueStudent(moduleTraitCor.HC,
nSamples.HC);

cell <- factor(rep(c("Bm1", "naive", "preGCB", "memory"), 6))
ID <- factor(c(rep(1, 4),rep(2, 4), rep(3, 4), rep(4, 4),rep(5,
4),rep(6, 4)))
MEG.HC <- cbind(MEs0.HC, cell, ID)

library(reshape2)
barplot(MEs0.SjS$MEblue, col="blue")
barplot(MEs0.SjS$MEbrown, col="brown")
barplot(MEs0.SjS$MEgreen, col="green")
barplot(MEs0.SjS$MEgrey, col="grey")
barplot(MEs0.SjS$MEturquoise, col="turquoise")
barplot(MEs0.SjS$MEyellow, col="yellow")

levels(MEG.SjS$cell)
```

```
MEG.SjS2  <-  transform(MEG.SjS,  cell=  factor(cell,  levels  =
c("Bm1", "naive", "preGCB", "memory")))

#SjS, blue
g <- ggplot(MEG.SjS2, aes(x = cell, y = MEblue, fill = ID))
g  <-  g  +  geom_bar(stat  =  "identity",  position  =  "dodge",
colour="black")
g    <-    g    +    theme_bw()    +    scale_fill_manual(values    =
c("blue","blue","blue","blue","blue","blue"), guide = "none")
gb  <-  g  +  ggtitle("SjS,  Module  Expression  Summary,  Blue")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gb)

#SjS, turquoise
g <- ggplot(MEG.SjS2, aes(x = cell, y = MEturquoise, fill = ID))
g  <-  g  +  geom_bar(stat  =  "identity",  position  =  "dodge",
colour="black")
g    <-    g    +    theme_bw()    +    scale_fill_manual(values    =
c("turquoise","turquoise","turquoise","turquoise","turquoise","
turquoise"), guide = "none")
gt <- g + ggtitle("SjS, Module Expression Summary, turquoise")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gt)

#SjS, yellow
g <- ggplot(MEG.SjS2, aes(x = cell, y = MEyellow, fill = ID))
g  <-  g  +  geom_bar(stat  =  "identity",  position  =  "dodge",
colour="black")
g    <-    g    +    theme_bw()+    scale_fill_manual(values    =
c("yellow","yellow","yellow","yellow","yellow","yellow"), guide
= "none")
gy  <-  g  +  ggtitle("SjS,  Module  Expression  Summary,  yellow")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gy)



#SjS, green
g <- ggplot(MEG.SjS2, aes(x = cell, y = MEgreen, fill = ID))
g <- g + geom_bar(stat = "identity", position = "dodge",
colour="black")
g <- g + theme_bw()+ scale_fill_manual(values =
c("green","green","green","green","green","green"), guide =
"none")
gg <- g + ggtitle("SjS, Module Expression Summary, green")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gg)

#SjS, brown
g <- ggplot(MEG.SjS2, aes(x = cell, y = MEbrown, fill = ID))
g <- g + geom_bar(stat = "identity", position = "dodge",
colour="black")
g <- g + theme_bw()+ scale_fill_manual(values =
c("brown","brown","brown","brown","brown","brown"), guide =
"none")
gbr <- g + ggtitle("SjS, Module Expression Summary, brown")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gbr)

#SjS, grey
g <- ggplot(MEG.SjS2, aes(x = cell, y = MEgrey, fill = ID))
g <- g + geom_bar(stat = "identity", position = "dodge",
colour="black")
```

```
g    <-    g    +    theme_bw()+    scale_fill_manual(values    =
c("grey","grey","grey","grey","grey","grey"), guide = "none")
ggr  <-  g  +  ggtitle("SjS,  Module  Expression  Summary,  grey")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(ggr)

library("gridExtra")
grid.arrange(gb, gt,gy,gg,gbr,ggr,ncol = 3)



levels(MEG.HC$cell)
MEG.HC2 <- transform(MEG.HC, cell= factor(cell, levels = c("Bm1",
"naive", "preGCB", "memory")))
#HC, blue
g <- ggplot(MEG.HC2, aes(x = cell, y = MEblue, fill = ID))
g  <-  g  +  geom_bar(stat  =  "identity",  position  =  "dodge",
colour="black")
g    <-    g    +    theme_bw()    +    scale_fill_manual(values    =
c("blue","blue","blue","blue","blue","blue"), guide = "none")
gb  <-  g  +  ggtitle("HC,  Module  Expression  Summary,  Blue")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gb)

#HC, turquoise
g <- ggplot(MEG.HC2, aes(x = cell, y = MEturquoise, fill = ID))
g  <-  g  +  geom_bar(stat  =  "identity",  position  =  "dodge",
colour="black")
g    <-    g    +    theme_bw()    +    scale_fill_manual(values    =
c("turquoise","turquoise","turquoise","turquoise","turquoise","
turquoise"), guide = "none")
gt <- g + ggtitle("HC, Module Expression Summary, turquoise")+
theme(axis.text=element_text(size=15),
axis.title=element_text(size=15,face="bold"))
plot(gt)

#HC, yellow
g <- ggplot(MEG.HC2, aes(x = cell, y = MEyellow, fill = ID))
g <- g + geom_bar(stat = "identity", position = "dodge",
colour="black")
g <- g + theme_bw()+ scale_fill_manual(values =
c("yellow","yellow","yellow","yellow","yellow","yellow"), guide
= "none")
gy <- g + ggtitle("HC, Module Expression Summary, yellow")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gy)



#HC, brown
g <- ggplot(MEG.HC2, aes(x = cell, y = MEbrown, fill = ID))
g <- g + geom_bar(stat = "identity", position = "dodge",
colour="black")
g <- g + theme_bw()+ scale_fill_manual(values =
c("brown","brown","brown","brown","brown","brown"), guide =
"none")
gbr <- g + ggtitle("HC, Module Expression Summary, brown")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(gbr)

#HC, grey
g <- ggplot(MEG.HC2, aes(x = cell, y = MEgrey, fill = ID))
g <- g + geom_bar(stat = "identity", position = "dodge",
colour="black")
g <- g + theme_bw()+ scale_fill_manual(values =
c("grey","grey","grey","grey","grey","grey"), guide = "none")
ggr <- g + ggtitle("HC, Module Expression Summary, grey")+
theme(axis.text=element_text(size=15),

axis.title=element_text(size=15,face="bold"))
plot(ggr)

library("gridExtra")
# πü╛πü¿πéüπüª1µ₧Üπü½σç║σè¢
grid.arrange(gb, gt,gy,ggr,gbr,ncol = 3)




#===============================================================
=======================
#
#  Code chunk 3
#
#===============================================================
=======================



sizeGrWindow(10,6)
trait <- c("CRP","Lym","IgG","SS.A","SS.B","ESSDAI")
moduleTraitCor.SjS2 <- moduleTraitCor.SjS[,trait]
moduleTraitPvalue.SjS2 <- moduleTraitPvalue.SjS[,trait]
# Will display correlations and their p-values
textMatrix.SjS =  paste(signif(moduleTraitCor.SjS2, 2), "¥n(",
                    signif(moduleTraitPvalue.SjS2, 1), ")", sep
= "");
dim(textMatrix.SjS) = dim(moduleTraitCor.SjS2)
par(mar = c(10, 12, 3, 3));
# Display the correlation values within a heatmap plot
datTraits2 <- datTraits[,trait]
rownames(moduleTraitCor.SjS2)
labeledHeatmap(Matrix = moduleTraitCor.SjS2,
            xLabels = names(datTraits2),
            yLabels = rownames(moduleTraitCor.SjS2),
            ySymbols = rownames(moduleTraitCor.SjS2),
            colorLabels = FALSE,
            colors = blueWhiteRed(50),
            textMatrix = textMatrix.SjS,
            setStdMargins = FALSE,
            cex.text = 1.8,cex.lab.x = 2,cex.lab.y = 2,
            zlim = c(-1,1),
            main = paste("SjS-Module-trait relationships"))


#===============================================================
=======================
#
#  Code chunk 4
#
#===============================================================
=======================



# Define variable weight containing the weight column of datTrait
ESSDAI = as.data.frame(datTraits$ESSDAI);
names(ESSDAI) = "ESSDAI"
# names (colors) of the modules
modNames.SjS = substring(names(MEs.SjS), 3)


geneModuleMembership.SjS    =    as.data.frame(cor(datExpr.SjS,
MEs.SjS, use = "p"));
MMPvalue.SjS                                                   =
as.data.frame(corPvalueStudent(as.matrix(geneModuleMembership.S
jS), nSamples.SjS));

names(geneModuleMembership.SjS)  =  paste("MM",  modNames.SjS,
sep="");
```

```
names(MMPvalue.SjS) = paste("p.MM.SjS", modNames.SjS, sep="");

geneTraitSignificance.SjS    =    as.data.frame(cor(datExpr.SjS,
ESSDAI, use = "p"));
GSPvalue.SjS                                                    =
as.data.frame(corPvalueStudent(as.matrix(geneTraitSignificance.
SjS), nSamples.SjS));

names(geneTraitSignificance.SjS) = paste("GS.", names(ESSDAI),
sep="");
names(GSPvalue.SjS) = paste("p.GS.", names(ESSDAI.SjS), sep="");


#================================================================
=======================
#
#  Code chunk 5
#
#================================================================
=======================



module.SjS = "grey"
column.SjS = match(module.SjS, modNames.SjS);
moduleGenes.SjS = moduleColors.SjS==module.SjS;

sizeGrWindow(7, 7);
par(mfrow = c(1,1));
verboseScatterplot(abs(geneModuleMembership.SjS[moduleGenes.SjS,
column.SjS]),
                abs(geneTraitSignificance.SjS[moduleGenes.SjS,
1]),
                xlab = paste("Module Membership in", module.SjS,
"module.SjS"),
                ylab = "Gene significance for ESSDAI",
                main   =   paste("Module   membership   vs.   gene
significance¥n"),
                cex.main = 1.2, cex.lab = 1.2, cex.axis = 1.2,
col = module.SjS)

library("ggsignif")
ESSDAI_MM <- read.csv("module_trait.csv", header=TRUE)
head(ESSDAI_MM)
moduleColour                                                <-
as.character(levels(ESSDAI_MM$moduleColors.SjS))

ggplot(ESSDAI_MM,  aes(x  =  moduleColors.SjS,  y  =  GS.ESSDAI,
color=moduleColors.SjS)) +
  geom_boxplot() +
  geom_jitter()+ scale_color_manual(values = moduleColour)

ggplot(ESSDAI_MM,     aes(x=abs(GS.ESSDAI),     y=abs(MM.grey),
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,     nrow     =     2)+geom_smooth(method     =     "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,     aes(x=abs(GS.ESSDAI),     y=abs(MM.blue),
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,     nrow     =     2)+geom_smooth(method     =     "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,     aes(x=abs(GS.ESSDAI),     y=abs(MM.green),
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,     nrow     =     2)+geom_smooth(method     =     "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,     aes(x=abs(GS.ESSDAI),    y=abs(MM.yellow),
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,     nrow     =     2)+geom_smooth(method     =     "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,     aes(x=abs(GS.ESSDAI),     y=abs(MM.brown),
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,      nrow      =      2)+geom_smooth(method      =      "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,   aes(x=abs(GS.ESSDAI),   y=abs(MM.turquoise),
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,      nrow      =      2)+geom_smooth(method      =      "lm")+
scale_color_manual(values = moduleColour)

ggplot(ESSDAI_MM,         aes(x=GS.ESSDAI,         y=MM.yellow,
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,      nrow      =      2)+geom_smooth(method      =      "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,          aes(x=GS.ESSDAI,          y=MM.blue,
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,      nrow      =      2)+geom_smooth(method      =      "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,         aes(x=GS.ESSDAI,         y=MM.brown,
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,      nrow      =      2)+geom_smooth(method      =      "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,         aes(x=GS.ESSDAI,         y=MM.green,
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,      nrow      =      2)+geom_smooth(method      =      "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,          aes(x=GS.ESSDAI,          y=MM.grey,
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,      nrow      =      2)+geom_smooth(method      =      "lm")+
scale_color_manual(values = moduleColour)
ggplot(ESSDAI_MM,       aes(x=GS.ESSDAI,       y=MM.turquoise,
color=moduleColors.SjS))+
geom_point()+theme_classic()+facet_wrap(~ESSDAI_MM$moduleColors
.SjS,     nrow     =     2)+geom_smooth(method     =     "lm")+
scale_color_manual(values = moduleColour)

Grey <- subset(ESSDAI_MM, moduleColors.SjS=="grey")
Green <- subset(ESSDAI_MM, moduleColors.SjS=="green")
Blue <- subset(ESSDAI_MM, moduleColors.SjS=="blue")
Brown <- subset(ESSDAI_MM, moduleColors.SjS=="brown")
Yellow <- subset(ESSDAI_MM, moduleColors.SjS=="yellow")
Turquoise <- subset(ESSDAI_MM, moduleColors.SjS=="turquoise")

cor.test(abs(ESSDAI_MM$GS.ESSDAI),      abs(ESSDAI_MM$MM.grey),
method = "pearson")
cor.test(Grey$GS.ESSDAI, Grey$MM.grey, method = "pearson")
cor.test(Green$GS.ESSDAI, Green$MM.green, method = "pearson")
cor.test(Blue$GS.ESSDAI, Blue$MM.blue, method = "pearson")
cor.test(abs(Green$GS.ESSDAI),  abs(Green$MM.brown),  method  =
"pearson")

ggplot(Yellow,  aes(x=Yellow$GS.ESSDAI,  y=Yellow$MM.yellow))  +
geom_point()
cor.test(Yellow$GS.ESSDAI, Yellow$MM.yellow, method = "pearson")
#===============================================================
=======================
#
#  Code chunk 6
#
#===============================================================
=======================


names(datExpr.SjS)


#===============================================================
=======================
#
#  Code chunk 7
#
#================================================================
=======================


names(datExpr.SjS)[moduleColors.SjS=="yellow"]


#================================================================
=======================
#
#  Code chunk 9
#
#================================================================
=======================


# Create the starting data frame
cell <- factor(rep(c("Bm1", "naive", "preGCB", "memory"), 4))
status <- factor(c(rep("HC", 24), rep("pSS", 24)))
ID <- factor(1:48)
Gene.Symbol <- datExpr.SjS$Gene_Symbol

##SjS
dim(datExpr.SjS)
probes = names(datExpr.SjS)
dim(x)

geneInfo0 = data.frame(probe = probes,
                   geneSymbol = Gene.Symbol,
                   moduleColors.SjS = moduleColors.SjS,
                   geneTraitSignificance.SjS,
                   GSPvalue.SjS)
```

```
head(geneInfo0)
geneInfo0$
  # Order modules by their significance for ESSDAI
  modOrder = order(-abs(cor(MEs.SjS, ESSDAI, use = "p")));
# Add module membership information in the chosen order
for (mod in 1:ncol(geneModuleMembership.SjS))
{
  oldNames = names(geneInfo0)
  geneInfo0 = data.frame(geneInfo0, geneModuleMembership.SjS[, 
modOrder[mod]],
                         MMPvalue.SjS[, modOrder[mod]]);
  names(geneInfo0)       =       c(oldNames,       paste("MM.", 
modNames.SjS[modOrder[mod]], sep=""),
                       paste("p.MM.",  modNames.SjS[modOrder[mod]], 
sep=""))
}
# Order the genes in the geneInfo variable first by module color, 
then by geneTraitSignificance
geneOrder      =      order(geneInfo0$moduleColors.SjS,      -
abs(geneInfo0$GS.ESSDAI));
geneInfo = geneInfo0[geneOrder, ]



##HC
dim(datExpr.HC)
probes = names(datExpr.HC)
dim(x)

geneInfo0.HC = data.frame(probe = probes,
                          geneSymbol = x$Gene.Symbol,
                          moduleColors.HC = moduleColors.HC)
head(geneInfo0.HC)
dim(geneInfo0.HC)
str(geneInfo0.HC)

#==============================================================
=======================
#
#  Code chunk 10
#
#=================================================================
=======================

#SjS
write.csv(geneInfo, file = "After_geneInfo.csv")

#HC
write.csv(geneInfo0.HC, file = "geneInfo.HC.csv")




##Network visualization using WGCNA functions

#=================================================================
=======================
#
#  Code chunk 2
#
#=================================================================
=======================

##SjS
# Calculate topological overlap anew: this could be done more
efficiently by saving the TOM
# calculated during module detection, but let us do it again
here.
dissTOM.SjS = 1-TOMsimilarityFromExpr(datExpr0.SjS, power = 6);
# Transform dissTOM with a power to make moderately strong
connections more visible in the heatmap
plotTOM.SjS = dissTOM.SjS^7;
# Set diagonal to NA for a nicer plot
```

```
diag(plotTOM.SjS) = NA;
# Call the plot function
sizeGrWindow(9,9)
TOMplot(plotTOM.SjS, geneTree.SjS, moduleColors.SjS, main =
"SjS_Network heatmap plot, all genes")


##HC
# Calculate topological overlap anew: this could be done more
efficiently by saving the TOM
# calculated during module detection, but let us do it again
here.
dissTOM.HC = 1-TOMsimilarityFromExpr(datExpr0.HC, power = 6);
# Transform dissTOM with a power to make moderately strong
connections more visible in the heatmap
plotTOM.HC = dissTOM.HC^7;
# Set diagonal to NA for a nicer plot
diag(plotTOM.HC) = NA;
# Call the plot function
sizeGrWindow(9,9)
TOMplot(plotTOM.HC, geneTree.HC, moduleColors.HC, main =
"HC_Network heatmap plot, all genes")


#================================================================
=======================
#
#  Code chunk 3
#
#================================================================
=======================


##SjS
nSelect = 400
# For reproducibility, we set the random seed
```

```
set.seed(10);
select.SjS = sample(nGenes.SjS, size = nSelect);
selectTOM.SjS = dissTOM.SjS[select.SjS, select.SjS];
# There's no simple way of restricting a clustering tree to a
subset of genes, so we must re-cluster.
selectTree.SjS  =  hclust(as.dist(selectTOM.SjS),  method  =
"average")
selectColors.SjS = moduleColors.SjS[select.SjS];
# Open a graphical window
sizeGrWindow(9,9)
# Taking the dissimilarity to a power, say 10, makes the plot
more informative by effectively changing
# the color palette; setting the diagonal to NA also improves
the clarity of the plot
plotDiss.SjS = selectTOM.SjS^7;
diag(plotDiss.SjS) = NA;
TOMplot(plotDiss.SjS, selectTree.SjS, selectColors.SjS, main =
"SjS.Network heatmap plot, selected genes")

##HC
nSelect = 400
# For reproducibility, we set the random seed
set.seed(10);
select.HC = sample(nGenes.HC, size = nSelect);
selectTOM.HC = dissTOM.HC[select.HC, select.HC];
# There's no simple way of restricting a clustering tree to a
subset of genes, so we must re-cluster.
selectTree.HC  =  hclust(as.dist(selectTOM.HC),  method  =
"average")
selectColors.HC = moduleColors.HC[select.HC];
# Open a graphical window
sizeGrWindow(9,9)
# Taking the dissimilarity to a power, say 10, makes the plot
more informative by effectively changing
# the color palette; setting the diagonal to NA also improves
the clarity of the plot
```

```
plotDiss.HC = selectTOM.HC^7;
diag(plotDiss.HC) = NA;
TOMplot(plotDiss.HC,  selectTree.HC,  selectColors.HC,  main  =
"HC.Network heatmap plot, selected genes")


#=================================================================
=======================
#
#  Code chunk 4
#
#=================================================================
=======================

##SjS
# Recalculate module eigengenes
MEs.SjS              =              moduleEigengenes(datExpr.SjS,
moduleColors.SjS)$eigengenes
# Isolate ESSDAI from the clinical traits
ESSDAI = as.data.frame(datTraits$ESSDAI);
names(ESSDAI) = "ESSDAI"
# Add the ESSDAI to existing module eigengenes
MET.SjS = orderMEs(cbind(MEs.SjS, ESSDAI))
# Plot the relationships among the eigengenes and the trait
sizeGrWindow(5,7.5);
par(cex = 0.9)
plotEigengeneNetworks(MET.SjS,  "",  marDendro  =  c(0,4,1,2),
marHeatmap = c(3,4,1,2), cex.lab = 0.8, xLabelsAngle
                  = 90)


#=================================================================
=======================
#
#  Code chunk 5
#
```

```
#=====================================================================
======================= 


# Plot the dendrogram
sizeGrWindow(6,6);
par(cex = 1.0)
plotEigengeneNetworks(MET.SjS,     "Eigengene     dendrogram",
marDendro = c(0,4,2,0),
                   plotHeatmaps = FALSE)
# Plot the heatmap matrix (note: this plot will overwrite the
dendrogram plot)
par(cex = 1.0)
plotEigengeneNetworks(MET.SjS,  "Eigengene  adjacency  heatmap",
marHeatmap = c(3,4,2,2),
                   plotDendrograms = FALSE, xLabelsAngle = 90)




##Export of networks to external software

#=====================================================================
=======================
#
#  Code chunk 2
#
#=====================================================================
=======================

# Recalculate topological overlap
TOM.SjS = TOMsimilarityFromExpr(datExpr.SjS, power = 6);
# Select module
module = "brown";
# Select module probes
probes.SjS = names(datExpr.SjS)
```

```
inModule.SjS = (moduleColors.SjS==module);
modProbes.SjS = probes.SjS[inModule.SjS];
# Select the corresponding Topological Overlap
modTOM.SjS = TOM.SjS[inModule.SjS, inModule.SjS];
dimnames(modTOM.SjS) = list(modProbes.SjS, modProbes.SjS)
# Export the network into an edge list file VisANT can read
vis.SjS = exportNetworkToVisANT(modTOM.SjS,
                           file = paste("VisANTInput-", module,
".txt", sep=""),
                           weighted = TRUE,
                           threshold = 0,
                           probeToGene  =  data.frame(y$X...ID,
y$Gene.Symbol) )


#================================================================
======================
#
#  Code chunk 3
#
#================================================================
======================

module = "brown";
nTop = 10;
IMConn.SjS = softConnectivity(datExpr.SjS[, modProbes.SjS]);
top.SjS = (rank(-IMConn.SjS) <= nTop)
vis.SjS = exportNetworkToVisANT(modTOM.SjS[top.SjS, top.SjS],
                           file = paste("VisANTInput-", module,
"-top30.txt", sep=""),
                           weighted = TRUE,
                           threshold = 0)


#================================================================
======================
#
#  Code chunk 4
#
#==============================================================
=======================


##SjS
#yellow
# Recalculate topological overlap if needed
TOM.SjS = TOMsimilarityFromExpr(datExpr.SjS, power = 6);
# Select modules
modules = c("yellow");
# Select module probes
probes.SjS = names(datExpr.SjS)
inModule.SjS = is.finite(match(moduleColors.SjS, modules));
modProbes.SjS = probes[inModule.SjS];
modGenes.SjS = y$Gene.Symbol[match(modProbes.SjS, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.SjS = TOM.SjS[inModule.SjS, inModule.SjS];
dimnames(modTOM.SjS) = list(modProbes.SjS, modProbes.SjS)
# Export the network into edge and node list files Cytoscape can
read
cyt.SjS.yellow = exportNetworkToCytoscape(modTOM.SjS,
                                   edgeFile                =
paste("SjS.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                   nodeFile                =
paste("SjS.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                   weighted = TRUE,
                                   threshold = 0.151,
                                   nodeNames = modProbes.SjS,
                                   altNodeNames = modGenes.SjS,
                                   nodeAttr                =
moduleColors.SjS[inModule.SjS])
```

```
str(cyt.SjS.yellow)

#blue
# Recalculate topological overlap if needed
TOM.SjS = TOMsimilarityFromExpr(datExpr.SjS, power = 6);
# Select modules
modules = c("blue");
# Select module probes
probes.SjS = names(datExpr.SjS)
inModule.SjS = is.finite(match(moduleColors.SjS, modules));
modProbes.SjS = probes[inModule.SjS];
modGenes.SjS = y$Gene.Symbol[match(modProbes.SjS, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.SjS = TOM.SjS[inModule.SjS, inModule.SjS];
dimnames(modTOM.SjS) = list(modProbes.SjS, modProbes.SjS)
# Export the network into edge and node list files Cytoscape can
read
cyt.SjS.blue = exportNetworkToCytoscape(modTOM.SjS,
                                  edgeFile                     =
paste("SjS.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                  nodeFile                     =
paste("SjS.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                  weighted = TRUE,
                                  threshold = 0.226,
                                  nodeNames = modProbes.SjS,
                                  altNodeNames = modGenes.SjS,
                                  nodeAttr                     =
moduleColors.SjS[inModule.SjS])
str(cyt.SjS.blue)

#brown
# Recalculate topological overlap if needed
TOM.SjS = TOMsimilarityFromExpr(datExpr.SjS, power = 6);
# Select modules
```

```
modules = c("brown");
# Select module probes
probes.SjS = names(datExpr.SjS)
inModule.SjS = is.finite(match(moduleColors.SjS, modules));
modProbes.SjS = probes[inModule.SjS];
modGenes.SjS = y$Gene.Symbol[match(modProbes.SjS, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.SjS = TOM.SjS[inModule.SjS, inModule.SjS];
dimnames(modTOM.SjS) = list(modProbes.SjS, modProbes.SjS)
# Export the network into edge and node list files Cytoscape can
read
cyt.SjS.brown = exportNetworkToCytoscape(modTOM.SjS,
                                 edgeFile                    =
paste("SjS.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                 nodeFile                    =
paste("SjS.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                 weighted = TRUE,
                                 threshold = 0.276,
                                 nodeNames = modProbes.SjS,
                                 altNodeNames = modGenes.SjS,
                                 nodeAttr                    =
moduleColors.SjS[inModule.SjS])
str(cyt.SjS.brown)

#green
# Recalculate topological overlap if needed
TOM.SjS = TOMsimilarityFromExpr(datExpr.SjS, power = 6);
# Select modules
modules = c("green");
# Select module probes
probes.SjS = names(datExpr.SjS)
inModule.SjS = is.finite(match(moduleColors.SjS, modules));
modProbes.SjS = probes[inModule.SjS];
modGenes.SjS = y$Gene.Symbol[match(modProbes.SjS, y$X...ID)];
```

```
# Select the corresponding Topological Overlap
modTOM.SjS = TOM.SjS[inModule.SjS, inModule.SjS];
dimnames(modTOM.SjS) = list(modProbes.SjS, modProbes.SjS)
# Export the network into edge and node list files Cytoscape can
read
cyt.SjS.green = exportNetworkToCytoscape(modTOM.SjS,
                                  edgeFile                    =
paste("SjS.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                  nodeFile                    =
paste("SjS.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                  weighted = TRUE,
                                  threshold = 0.148,
                                  nodeNames = modProbes.SjS,
                                  altNodeNames = modGenes.SjS,
                                  nodeAttr                    =
moduleColors.SjS[inModule.SjS])
str(cyt.SjS.green)


#grey
# Recalculate topological overlap if needed
TOM.SjS = TOMsimilarityFromExpr(datExpr.SjS, power = 6);
# Select modules
modules = c("grey");
# Select module probes
probes.SjS = names(datExpr.SjS)
inModule.SjS = is.finite(match(moduleColors.SjS, modules));
modProbes.SjS = probes[inModule.SjS];
modGenes.SjS = y$Gene.Symbol[match(modProbes.SjS, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.SjS = TOM.SjS[inModule.SjS, inModule.SjS];
dimnames(modTOM.SjS) = list(modProbes.SjS, modProbes.SjS)
# Export the network into edge and node list files Cytoscape can
read
```

```
cyt.SjS.grey = exportNetworkToCytoscape(modTOM.SjS,
                                 edgeFile                    =
paste("SjS.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                 nodeFile                    =
paste("SjS.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                 weighted = TRUE,
                                 threshold = 0.035,
                                 nodeNames = modProbes.SjS,
                                 altNodeNames = modGenes.SjS,
                                 nodeAttr                    =
moduleColors.SjS[inModule.SjS])
str(cyt.SjS.grey)

#turquoise
# Recalculate topological overlap if needed
TOM.SjS = TOMsimilarityFromExpr(datExpr.SjS, power = 6);
# Select modules
modules = c("turquoise");
# Select module probes
probes.SjS = names(datExpr.SjS)
inModule.SjS = is.finite(match(moduleColors.SjS, modules));
modProbes.SjS = probes[inModule.SjS];
modGenes.SjS = y$Gene.Symbol[match(modProbes.SjS, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.SjS = TOM.SjS[inModule.SjS, inModule.SjS];
dimnames(modTOM.SjS) = list(modProbes.SjS, modProbes.SjS)
# Export the network into edge and node list files Cytoscape can
read
cyt.SjS.turquoise = exportNetworkToCytoscape(modTOM.SjS,
                                     edgeFile                =
paste("SjS.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                     nodeFile                =
paste("SjS.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                      weighted = TRUE,
                                      threshold = 0.461,
                                      nodeNames = modProbes.SjS,
                                      altNodeNames               =
modGenes.SjS,
                                      nodeAttr                   =
moduleColors.SjS[inModule.SjS])
str(cyt.SjS.turquoise)




##HC
#yellow
# Recalculate topological overlap if needed
TOM.HC = TOMsimilarityFromExpr(datExpr.HC, power = 6);
# Select modules
modules = c("yellow");
# Select module probes
probes.HC = names(datExpr.HC)
inModule.HC = is.finite(match(moduleColors.HC, modules));
modProbes.HC = probes[inModule.HC];
modGenes.HC = y$Gene.Symbol[match(modProbes.HC, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.HC = TOM.HC[inModule.HC, inModule.HC];
dimnames(modTOM.HC) = list(modProbes.HC, modProbes.HC)
# Export the network into edge and node list files Cytoscape can
read
cyt.HC.yellow = exportNetworkToCytoscape(modTOM.HC,
                                  edgeFile                      =
paste("HC.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                  nodeFile                      =
paste("HC.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                  weighted = TRUE,
                                  threshold = 0.31,
                                  nodeNames = modProbes.HC,
                                  altNodeNames = modGenes.HC,
                                  nodeAttr                    =
moduleColors.HC[inModule.HC])
str(cyt.HC.yellow)

#blue
# Recalculate topological overlap if needed
TOM.HC = TOMsimilarityFromExpr(datExpr.HC, power = 6);
# Select modules
modules = c("blue");
# Select module probes
probes.HC = names(datExpr.HC)
inModule.HC = is.finite(match(moduleColors.HC, modules));
modProbes.HC = probes[inModule.HC];
modGenes.HC = y$Gene.Symbol[match(modProbes.HC, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.HC = TOM.HC[inModule.HC, inModule.HC];
dimnames(modTOM.HC) = list(modProbes.HC, modProbes.HC)
# Export the network into edge and node list files Cytoscape can
read
cyt.HC.blue = exportNetworkToCytoscape(modTOM.HC,
                                edgeFile                      =
paste("HC.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                nodeFile                      =
paste("HC.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                weighted = TRUE,
                                threshold = 0.365,
                                nodeNames = modProbes.HC,
                                altNodeNames = modGenes.HC,
                                nodeAttr                      =
moduleColors.HC[inModule.HC])
str(cyt.HC.blue)

#brown
# Recalculate topological overlap if needed
TOM.HC = TOMsimilarityFromExpr(datExpr.HC, power = 6);
# Select modules
modules = c("brown");
# Select module probes
probes.HC = names(datExpr.HC)
inModule.HC = is.finite(match(moduleColors.HC, modules));
modProbes.HC = probes[inModule.HC];
modGenes.HC = y$Gene.Symbol[match(modProbes.HC, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.HC = TOM.HC[inModule.HC, inModule.HC];
dimnames(modTOM.HC) = list(modProbes.HC, modProbes.HC)
# Export the network into edge and node list files Cytoscape can
read
cyt.HC.brown = exportNetworkToCytoscape(modTOM.HC,
                                 edgeFile              =
paste("HC.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                 nodeFile              =
paste("HC.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                 weighted = TRUE,
                                 threshold = 0.318,
                                 nodeNames = modProbes.HC,
                                 altNodeNames = modGenes.HC,
                                 nodeAttr              =
moduleColors.HC[inModule.HC])
str(cyt.HC.brown)


#grey
# Recalculate topological overlap if needed
```

```
TOM.HC = TOMsimilarityFromExpr(datExpr.HC, power = 6);
# Select modules
modules = c("grey");
# Select module probes
probes.HC = names(datExpr.HC)
inModule.HC = is.finite(match(moduleColors.HC, modules));
modProbes.HC = probes[inModule.HC];
modGenes.HC = y$Gene.Symbol[match(modProbes.HC, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.HC = TOM.HC[inModule.HC, inModule.HC];
dimnames(modTOM.HC) = list(modProbes.HC, modProbes.HC)
# Export the network into edge and node list files Cytoscape can
read
cyt.HC.grey = exportNetworkToCytoscape(modTOM.HC,
                                edgeFile                     =
paste("HC.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                nodeFile                     =
paste("HC.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                weighted = TRUE,
                                threshold = 0.01,
                                nodeNames = modProbes.HC,
                                altNodeNames = modGenes.HC,
                                nodeAttr                     =
moduleColors.HC[inModule.HC])
str(cyt.HC.grey)

#turquoise
# Recalculate topological overlap if needed
TOM.HC = TOMsimilarityFromExpr(datExpr.HC, power = 6);
# Select modules
modules = c("turquoise");
# Select module probes
probes.HC = names(datExpr.HC)
inModule.HC = is.finite(match(moduleColors.HC, modules));
```

```
modProbes.HC = probes[inModule.HC];
modGenes.HC = y$Gene.Symbol[match(modProbes.HC, y$X...ID)];
# Select the corresponding Topological Overlap
modTOM.HC = TOM.HC[inModule.HC, inModule.HC];
dimnames(modTOM.HC) = list(modProbes.HC, modProbes.HC)
# Export the network into edge and node list files Cytoscape can
read
cyt.HC.turquoise = exportNetworkToCytoscape(modTOM.HC,
                                     edgeFile                  =
paste("HC.CytoscapeInput-edges-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                     nodeFile                  =
paste("HC.CytoscapeInput-nodes-", paste(modules, collapse="-"),
"-.txt", sep=""),
                                     weighted = TRUE,
                                     threshold = 0.5,
                                     nodeNames = modProbes.HC,
                                     altNodeNames = modGenes.HC,
                                     nodeAttr                  =
moduleColors.HC[inModule.HC])
str(cyt.HC.turquoise)
```