Tutorial on studying module preservation: III. Preservation of human brain modules in chimpanzee brains and vice versa

Peter Langfelder and Steve Horvath

October 25, 2010

Contents

1 Overview

In this tutorial we provide a detailed, step-by-step analysis of module preservation between human and chimpanzee brains. Unlike Tutorial I (preservation of female mouse liver modules in male samples) in this case module assignments are available for both data sets, allowing us to use the standard cross-tabulation statistics in addition to network preservation statistics.
The data and a large part of the network analysis used in this example have been described in [2]. The human and chimpanzee data sets contain 18 samples each. The samples come from 6 matched brain regions; thus, each region is represented by 3 samples. We repeat the network analysis, and add an analogous network analysis and module detection for the chimp data, in a separate document.
Here we simply load the expression data and module labels, call the function `modulePreservation` that calculates a broad range of preservation statistics, and analyze its output. We also provide code that we used to create figures for the main paper. We also evaluate the performance of In-Group Proportion (IGP) [1].
We encourage readers unfamiliar with any of the functions used in this tutorial to open an R session and type

```
help(functionName)
```

(replace `functionName` with the actual name of the function) to get a detailed description of what the functions does, what the input arguments mean, and what is the output.

1.a Setting up the R session

After starting R we execute a few commands to set the working directory and load the requisite packages:

```
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load the package
library(WGCNA);
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
```

2 Data input

2.a Input of expression data

We assume the user has downloaded the data for this tutorial and saved them in a local directory. The expression data is contained in the file `Dataset 1 (network construction).csv.bz2`.

```
file = bzfile("Dataset 1 (network construction).csv.bz2");
dat1 = read.csv(file, header=T)
dim(dat1)
names(dat1);
```

In addition to expression data, the data files contain extra information about the surveyed genes we do not need. One can inspect larger data frames such as `dat1` by invoking R data editor via `fix(dat1)`. The expression data set contains 36 samples. Note that each row corresponds to a gene and column to a sample or auxiliary information. We now remove the auxiliary data and transpose the expression data for further analysis. Further, we only keep probesets marked with a number greater than zero in the column `Brain_variant_H`.

```
datExpr=data.frame(t(dat1[dat1$Brain_variant_H>0,2:39]))
indexHuman=c(19:36)
indexChimp=c(1:18)
```

We now set up the multi-set expression data and corresponing module colors:

```
# Number of data sets that we work with
nSets = 2;
# Object that will contain the expression data
multiExpr = list();
multiExpr[[1]] = list(data = datExpr[indexHuman, ]);
multiExpr[[2]] = list(data = datExpr[indexChimp, ]);
# Names for the two sets
setLabels = c("Human", "Chimp");
# Important: components of multiExpr must carry identificating names
names(multiExpr) = setLabels
# Display the dimensions of the expression data (if you are confused by this construct, ignore it):
lapply(multiExpr, lapply, dim)
```

The output of the last command basically confirms that we have two expression data sets, each of which contains expression measurements of 4000 genes in 18 samples. Note the structure of `multiExpr`: it is a list with one component per input data set. For each data set, the component is in turn another list that must contain a component called `data` and that contains the expression data. The outer list must have appropriate names so that the expression data can be matched to module labels that we create next.

2.b Loading of module labels

We load the module labels obtained from network analysis and create a list holding the module colors for each of the data sets.

```
x = load("HumanChimp-OldhamAnalysis-colorHuman-colorChimp-inNetwork.RData")
# Create an object (list) holding the module labels for each set:
colorList = list(colorHuman, colorChimp);
# Components of the list must be named so that the names can be matched to the names of multiExpr
names(colorList) = setLabels;
```

Again, note that the names of `multiExpr` and names of `multiColor` must correspond. We invite the reader to read through Appendix A to see how easy it is to construct the networks and identify modules.

3 Calculation of module preservation statistics

Roll the drums, here comes the calculation of module preservation. This calculation takes about an hour on a fast deskopt PC. If this code has been already executed once and the result is saved, it can be loaded from disk, thus saving time.

```
system.time( {
   mp = modulePreservation(multiExpr, colorList,
                          referenceNetworks = c(1:2),
                          loadPermutedStatistics = FALSE,
                          nPermutations = 200,
                          verbose = 3)
         } );
# Save the results
save(mp, file = "HumanChimp-HumanSpecific-modulePreservation.RData");
```

4 Calculation of In-Group Proportion

Here we use the function `clusterRepro` from the package `clusterRepro` to calculate IGP statistics of the human modules in chimp data and vice-versa. The package `clusterRepro` must be installed before running this code.

```
# Impute missing data and calculate eigengenes
impExpr = list();
for (set in 1:nSets)
{
  impExpr[[set]] = list(data = t(impute.knn(t(multiExpr[[set]]$data))$data));
}
eigengenes = list();
for (set in 1:nSets)
{
  eigengenes[[set]] = multiSetMEs(impExpr, universalColors = colorList[[set]], excludeGrey = TRUE);
  for (ss in 1:nSets)
  {
    rownames(eigengenes[[set]][[ss]]$data) = rownames(multiExpr[[ss]]$data);
  }
}
# Here comes the IGP calculation
library(clusterRepro)
cr = list();
set.seed(20);
for (ref in 1:nSets)
{
  cr[[ref]] = list();
  for (test in 1:nSets)
  {
    printFlush(system.time({
              cr[[ref]][[test]] = clusterRepro(Centroids = as.matrix(eigengenes[[ref]][[test]]$data),
                                    New.data = as.matrix(impExpr[[test]]$data),
                                    Number.of.permutations = 10000); }));
    collectGarbage();
  }
}
# Save the results
save(cr, file = "HumanChimp-HumanSpecific-clusterRepro.RData");
```

Once the IGP results have been calculated and saved, they can be re-used without spending days on the calculation.

5 Analysis and display of module preservation results

We now analyze the calculated module preservation statistics. Isolate the observed statistics and their Z scores:

```
# Load the module preservation statistics
load(file = "HumanChimp-HumanSpecific-clusterRepro.RData");
load(file = "HumanChimp-HumanSpecific-modulePreservation.RData")
ref = 1 # Select the human data as reference
test = 2 # Select the chimp data as test
statsObs = cbind(mp$quality$observed[[ref]][[test]][, -1], mp$preservation$observed[[ref]][[test]][, -1])
statsZ = cbind(mp$quality$Z[[ref]][[test]][, -1], mp$preservation$Z[[ref]][[test]][, -1]);
```

We look at the main output: the preservation Zsummary score.

```
print(signif(statsZ[, "Zsummary.pres", drop = FALSE],2));
# Compare preservation to quality:
print(signif(statsZ[, c("Zsummary.pres", "Zsummary.qual")], 2))
```

We invite the reader to switch the reference and test sets by setting `ref=2; test=1` and look at the output.

5.a Plots of summary statistics

The numbers are nice, but (to paraphrase a saying) a picture is worth a thousand numbers. We now produce Figure 4 of the main article in which we present $Z_{summary}$, *medianRank*, observed IGP, and IGP permutation p-value for the preservation of human modules in chimp brain expression data. We prepare several plotting variables so the actual plotting can be done within a for loop. We start by setting up the preservation statistics, module sizes etc.

```
ref = 1;
test = 2;
ind = 1;
stats= mp$preservation$observed[[ref]][[test]];
labelsX = rownames(stats)
labelsX[labelsX=="gold"] = "orange"
modColors = labelsX;
plotMods = !(modColors %in% c("grey", "orange"));
moduleSizes = stats[plotMods, 1];
textLabels = match(modColors, standardColors(20))[plotMods];
colorLabels = labelsX[plotMods];
```

Next we prepare a list containing the x and y coordinates of scatterplots and set up necessary options.

```
nModules = sum(plotMods);
nPlots = 6
plotData = list();
# Fill up the plotData
plotData[[1]] = plotData[[2]] = matrix(0, nModules, nPlots);
plotData[[1]][, c(1:4)] = moduleSizes;
plotData[[2]][, 1] = mp$preservation$Z[[ref]][[test]]$Zsummary.pres[plotMods];
plotData[[2]][, 2] = mp$preservation$observed[[ref]][[test]]$medianRank.pres[plotMods];
# Match the modulePreservation ordering of modules to that of clusterRepro
crLabels = sort(unique(colorLabels));
mp2cr = match(colorLabels, crLabels);
# Scatterplots of IGP and p-value vs. module size
plotData[[2]][, 3] = cr[[ref]][[test]]$Actual.IGP
plotData[[2]][, 4] = -log10(cr[[ref]][[test]]$p.value + 1e-4);
# Scatterplot of observed IGP vs. Zsummary and medianRank
plotData[[1]][, c(5,6)] = plotData[[2]][, c(1:2)];
plotData[[2]][, c(5,6)] = plotData[[2]][, 3];
# Plot annotation
xLabs = c(rep("Module size", 4), "Zsummary", "Median rank");
yLabs = c("Zsummary", "Median rank", "Observed IGP", "-log10(IGP perm p)", "Observed IGP", "Observed IGP");
mains = spaste(LETTERS[1:nPlots], ". ", #rep("Ref: Human, Test: Chimp\n", nPlots),
              c(yLabs[1:4], paste(yLabs[5:6], "vs.", xLabs[5:6])),
              c("", "", "", "", "\n", "\n"));
# Scatterplot options
verbose = c(rep(FALSE, 4), rep(TRUE, 2));
ablines = list(c(0, 2, 10), NA, NA, c(-log10(0.05), -log10(0.05/nModules)), NA, NA);
abColors = list(c("black", "blue", "darkgreen"), NA, NA, c("blue", "red"), NA, NA);
logs = c("x", "x", "x", "x", "", "");
invertY = c(FALSE, TRUE, rep(FALSE, 4));
verSP = function(...) { verboseScatterplot(..., abline = TRUE) }
```

The actual plotting code starts here.

```
cexLabels = 1.4
sizeGrWindow(6,9);
#pdf(file = "Plots/HumanSpecific-NetworkAndIGPStatistics.pdf", w=6, h=9, onefile = FALSE);
par(mfrow = c(3,2));
par(mar = c(3.3, 3.3, 3.2, 0.5));
par(mgp = c(2, 0.6, 0))
for (p in 1:nPlots)
{
  x = plotData[[1]][, p];
  y = plotData[[2]][, p]
  miny = min(y, ablines[[p]], na.rm = TRUE);
  maxy = max(y, ablines[[p]], na.rm = TRUE);
  miny = miny - (maxy-miny)*0.1;
  maxy = maxy + (maxy-miny)*0.1;
  (if (verbose[p]) verSP else plot ) (plotData[[1]][, p], plotData[[2]][, p],
                     main = mains[p],
                     xlab = xLabs[p],
                     ylab = yLabs[p],
                     cex.main = cexLabels, cex.lab = cexLabels, cex.axis = cexLabels,
                     bg = colorLabels,
                     col = colorLabels, cex = 2.2,
                     ylim = if (invertY[p]) c(maxy, miny) else c(miny, maxy),
                     pch = 21,
                     log = logs[p]);
  labelPoints(plotData[[1]][, p], plotData[[2]][, p], textLabels, cex = cexLabels, offs = 0.06);
  if (!is.na(ablines[[p]][[1]]))
    for (al in 1:length(ablines[[p]]))
      abline(h = ablines[[p]][[al]], col = abColors[[p]][[al]], lty = 2);
}
# If plotting into a pdf file, close the file. An un-closed pdf file is not readable.
dev.off();
```

The resulting plot is shown in Figure 1. We find that the green and blue module are the least preserved as identified by *medianRank*. The $Z_{summary}$ stastic is relatively high for all modules, meaning that they are all preserved at least to some degree.

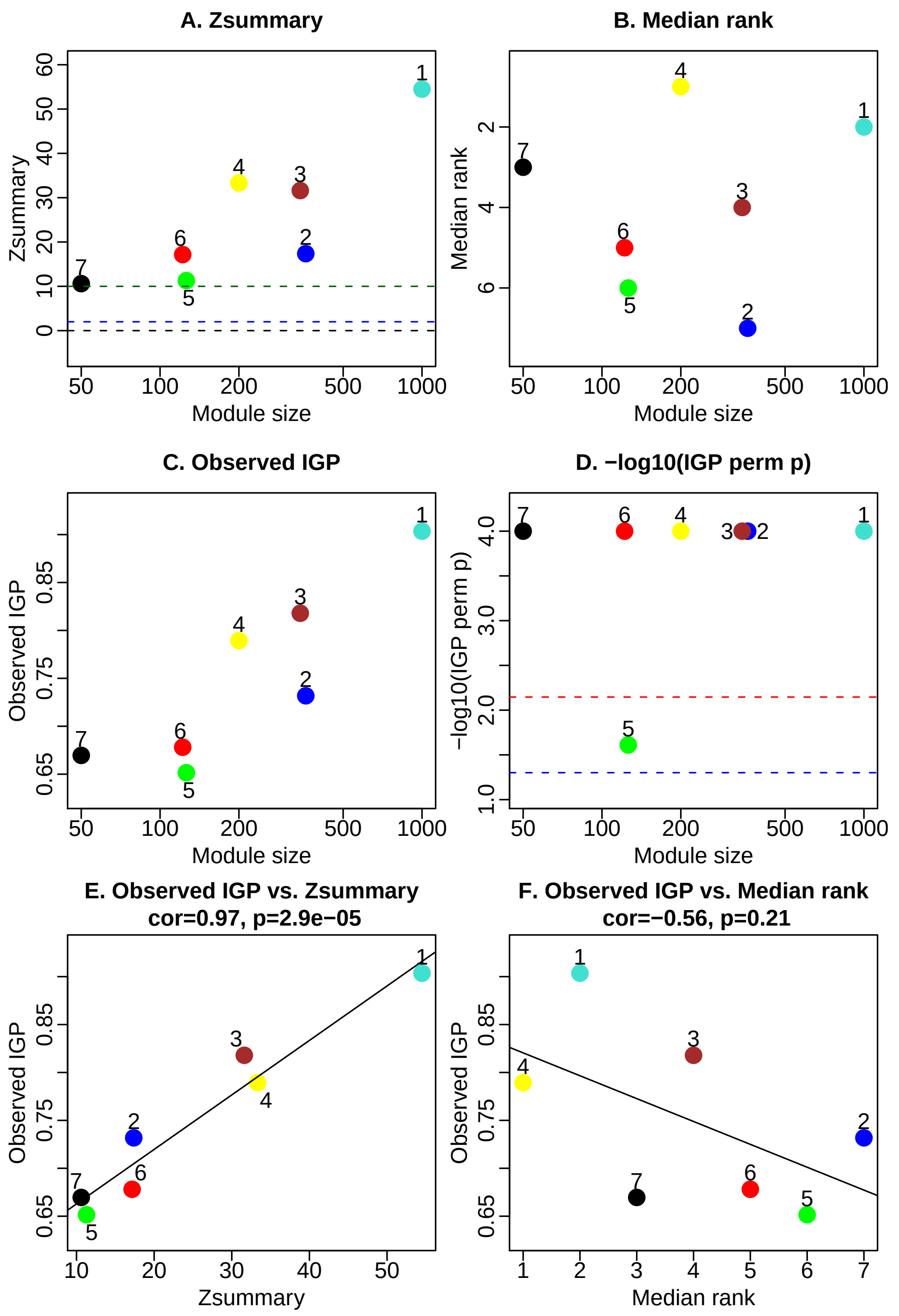

Figure 1: Composite preservation statistics of human modules in chimpanzee samples. A. The summary statistic $Z_{summary}$ (y-axis) as a function of the module size. Each point represents a module, labeled by color and a secondary numeric label (1=turquoise, 2=blue, 3=brown, 4=yellow, 5=green, 6=red, 7=black). The dashed blue and green lines indicate the thresholds $Z = 2$ and $Z = 10$, respectively. B. The composite statistic $medianRank$ (y-axis) as a function of the module size. Each point represents a module, labeled by color and a secondary numeric label as in panel A. Low numbers on the y axis indicate a high preservation. C. Observed IGP statistic (Kapp and Tibshirani, 2007) versus module size. D. P-value of the IGP statistic versus module size. E. and F. show scatter plots between the observed IGP statistic and $Z_{summary}$ and $medianRank$, respectively. In this example, where modules are defined as clusters, the IGP statistic has a high positive correlation ($r = 0.97$) with $Z_{summary}$ and a moderately large negative correlation ($r = -0.56$) with $medianRank$. The negative correlation is expected since low median ranks indicate high preservation.

5.b Plots of all statistics

We now plot all quality and preservation statistics into separate pdf files. These plots are included in Supplementary text S1along with the main paper. The purpose of these plots is to see the performance and results of individual quality and preservation statistics.

```
figNames = c("quality", "preservation");
useStats = list(c(1:8)+1, c(9:27)+1);
sectioning = list(c(2,4), c(4,5));
dims = list(sectioning[[1]] * 1.8, sectioning[[2]]*1.8);
for (f in 1:2)
{
  pdf(file=spaste("Plots/HumanChimp-HumanSpecific-modulePreservation-",figNames[f],"-%02d.pdf"),
      w=dims[[f]][2], h=dims[[f]][1], onefile = FALSE)

  for (ref in 1:nSets) for (test in 1:nSets) if (ref!=test)
  {
    stats = cbind(mp$quality$Z[[ref]][[test]],
                  mp$referenceSeparability$Z[[ref]][[test]][, -1, drop = FALSE],
                  mp$preservation$Z[[ref]][[test]][, -1],
                  mp$accuracy$Z[[ref]][[test]][, -1],
                  mp$testSeparability$Z[[ref]][[test]][, -1, drop = FALSE]);
    labelsX = rownames(stats)
    labelsX[labelsX=="gold"] = "orange"
    modColors = labelsX;
    moduleSizes = stats[, 1];
    plotMods = !(modColors %in% c("grey", "orange"));
    plotStats = useStats[[f]]
    textLabels = match(modColors, standardColors(20))[plotMods];

    par(mfrow = sectioning[[f]]);
    par(mar = c(3.0, 3.0, 4, 0.4));
    par(mgp = c(1.7, 0.6, 0))

    for (s in plotStats)
    {
       min = min(stats[plotMods, s], na.rm = TRUE);
       max = max(stats[plotMods, s], na.rm = TRUE);
       minMS = min(moduleSizes[plotMods]);
       maxMS = max(moduleSizes[plotMods]);
       nms = colnames(stats);

       if (max < 10) max = 10;

       if (min > -max/5) min = -max/5
       plot(moduleSizes[plotMods], stats[plotMods, s], col = 1, bg = modColors[plotMods], pch = 21,
            main = paste("Ref.:", setLabels[ref], "\nTest:", setLabels[test], "\n", nms[s]),
            cex = 1.2,
            cex.main = 1.0,
            ylab = nms[s], xlab = "Module size", log = "x",
            ylim = c(min, max + 0.1 * (max-min)),
            xlim = c(minMS/(maxMS/minMS)^0.1, maxMS*(maxMS/minMS)^0.1)
            )
       labelPoints(moduleSizes[plotMods], stats[plotMods, s], textLabels, cex = 0.90, offs = 0.06);
       abline(h=0)
       abline(h=2, col = "blue", lty = 2)
       abline(h=10, col = "darkgreen", lty = 2)
    }
  }
  dev.off();
}
```

We display one of the generated plots in Figure 2.

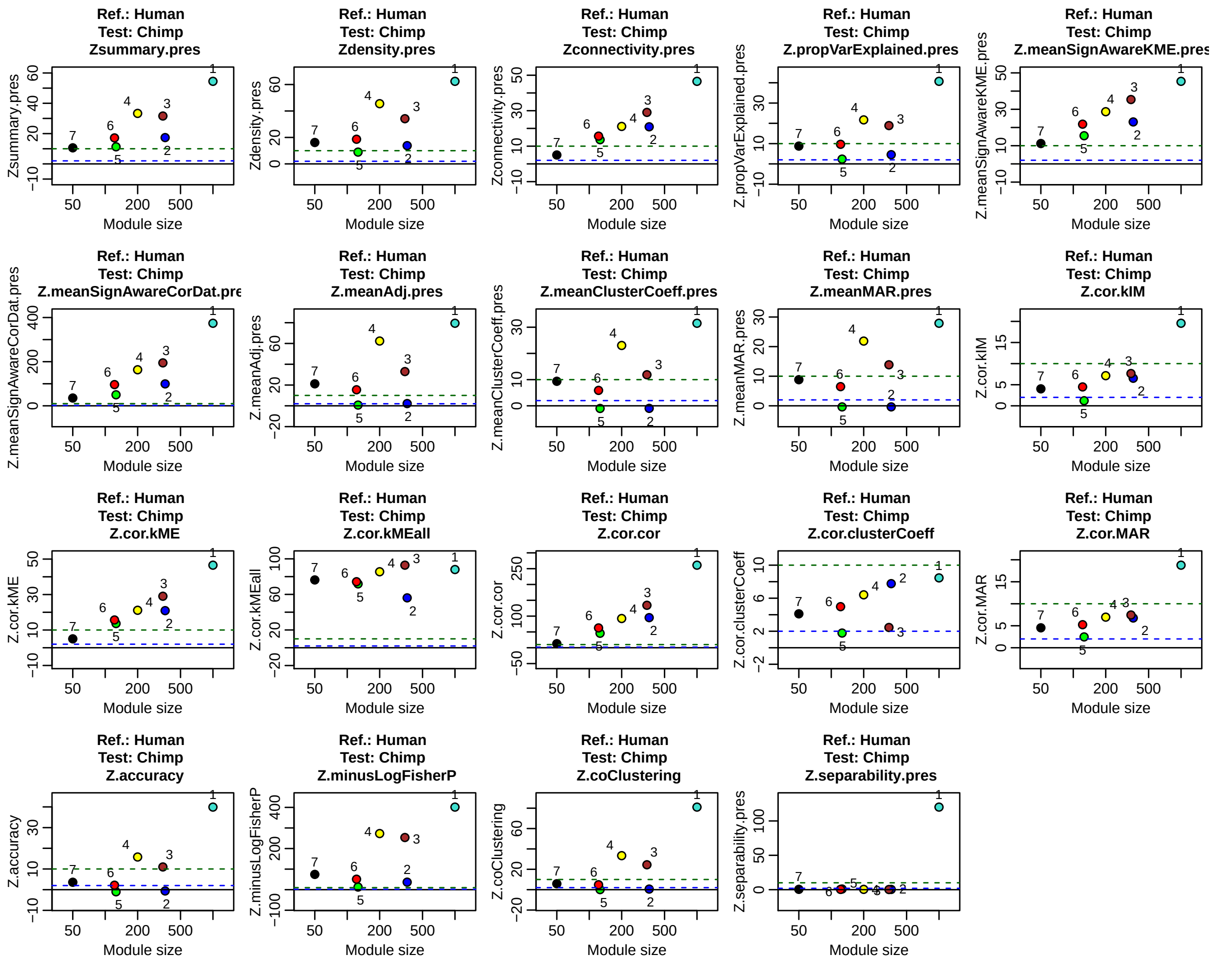

Figure 2: Permutation Z scores of module preservation statistics of human modules with respect to genes also present in the chimp samples. Each plot shows the permutation Z scores of a module preservation statistic (indicated in the tile) as a function of the module size. Dashed blue and green lines indicate the thresholds $Z = 2$ and $Z = 10$, respectively.

5.c Output of complete results

We now output an Excel sheet that contains all observed statistics and their permutation scores for this study. This is in principle straightforward, with only one subtlety: the accuracy statistics do not contain an entry for the "gold" module. The "gold" module consists of 1000 randomly selected genes that represent a sample of the whole network, and accuracy measures have no meaning for this module. To output all statistics together, we add missing values (`NA`) for the accuracy of the gold module.

```
# This variable will contain the summary table
summaryTable = NULL
# Loop over all combinations of reference and tests sets
for (ref in 1:nSets) for (test in 1:nSets) if (ref!=test)
{
  modules = rownames(mp$preservation$Z[[ref]][[test]]);
  nMods = length(modules);
  sizes = mp$preservation$Z[[ref]][[test]][, 1];
  acc = matrix(NA, nMods, 3);
  if (test!=4)
  {
    acc[match(rownames(mp$accuracy$observed[[ref]][[test]]), modules), ] =
             mp$accuracy$observed[[ref]][[test]][, -1, drop = FALSE];
    colnames(acc) = colnames(mp$accuracy$observed[[ref]][[test]])[-1];
    accZ = mp$accuracy$Z[[ref]][[test]][, -1, drop = FALSE];
    acc.log.p = mp$accuracy$log.p[[ref]][[test]][, -1, drop = FALSE];
    acc.log.pBonf = mp$accuracy$log.pBonf[[ref]][[test]][, -1, drop = FALSE];
  } else {
    accZ = matrix(NA, nMods, 3);
    acc.log.p = matrix(NA, nMods, 3);
    acc.log.pBonf = matrix(NA, nMods, 3);
    colnames(acc) = colnames(mp$accuracy$observed[[1]][[2]])[-1];
    colnames(accZ) = colnames(mp$accuracy$Z[[1]][[2]])[-1];
    colnames(acc.log.p) = colnames(mp$accuracy$log.p[[1]][[2]])[-1];
    colnames(acc.log.pBonf) = colnames(mp$accuracy$log.pBonf[[1]][[2]])[-1];
  }
  # Table of results for this reference-test combination
  tab = cbind(referenceSet = rep(setLabels[ref], nMods),
              testSet = rep(setLabels[test], nMods),
              moduleLabel = modules,
              moduleSize = sizes,
              mp$quality$observed[[ref]][[test]][, -1, drop = FALSE],
              mp$preservation$observed[[ref]][[test]][, -1, drop = FALSE],
              acc,
              mp$referenceSeparability$observed[[ref]][[test]][, -1, drop = FALSE],
              mp$testSeparability$observed[[ref]][[test]][, -1, drop = FALSE],
              mp$quality$Z[[ref]][[test]][, -1, drop = FALSE],
              mp$quality$log.p[[ref]][[test]][, -1, drop = FALSE],
              mp$quality$log.pBonf[[ref]][[test]][, -1, drop = FALSE],
              mp$preservation$Z[[ref]][[test]][, -1, drop = FALSE],
              mp$preservation$log.p[[ref]][[test]][, -1, drop = FALSE],
              mp$preservation$log.pBonf[[ref]][[test]][, -1, drop = FALSE],
              accZ,
              acc.log.p,
              acc.log.pBonf,
              mp$referenceSeparability$Z[[ref]][[test]][, -1, drop = FALSE],
              mp$referenceSeparability$log.p[[ref]][[test]][, -1, drop = FALSE],
              mp$referenceSeparability$log.pBonf[[ref]][[test]][, -1, drop = FALSE],
              mp$testSeparability$Z[[ref]][[test]][, -1, drop = FALSE],
              mp$testSeparability$log.p[[ref]][[test]][, -1, drop = FALSE],
              mp$testSeparability$log.pBonf[[ref]][[test]][, -1, drop = FALSE]
             )
  # Add the table to the main table.
  if (is.null(summaryTable)) summaryTable = tab else summaryTable = rbind(summaryTable, tab);
}
# Save the table in csv format.
write.table(summaryTable, file = "HumanChimp-HumanSpecific-completeResults.csv", row.names = FALSE,
            sep = ",", quote = FALSE);
```

The resulting table is included as Supplementary Table S2with the main paper.

5.d Summary figures used in the main paper

Here we provide the code that generates the summary figures for this application that were used in the main paper. Before we begin with the figures, we re-create the gene clustering dendrograms in both the human and the chimp data. Note the soft thresholding power is 9.

```
dendrograms = list();
for (set in 1:nSets)
{
  adj = abs(cor(multiExpr[[set]]$data[, inNetwork], use = "p"))^9;
  dtom = TOMdist(adj);
  dendrograms[[set]] = flashClust(as.dist(dtom), method = "a");
}
# Get eigengenes
mes = list()
for (set in 1:nSets)
{
   mes[[set]] = moduleEigengenes(multiExpr[[set]]$data, colorList[[ref]])$eigengenes
}
```

We now create a contingency (or overlap) table of human and chimp modules together with Fisher's exact test p-values for observing the observed overlaps by chance. The WGCNA packages contains a convenient function `overlapTable` that does the necessary calculations.

```
# Calculate the contingency table and p-values
overlap = overlapTable(colorHuman[inNetwork], colorChimp[inNetwork]);
# The numMat will encode color. We use -log of the p value.
numMat = -log10(overlap$pTable);
numMat[numMat >50] = 50;
# Prepare for generating a color-coded plot of the overlap table. The text of the table will consist of
# counts and corresponding p-values.
textMat = paste(overlap$countTable, "\n", signif(overlap$pTable, 2));
dim(textMat) = dim(numMat)
# Additional information for the plot. These will be used shortly.
xLabels = paste("M", sort(unique(colorChimp)));
yLabels = paste("M", sort(unique(colorHuman)));
xSymbols = paste(sort(unique(colorChimp)), ": ", table(colorChimp[inNetwork]), sep = "")
ySymbols = paste(sort(unique(colorHuman)), ": ", table(colorHuman[inNetwork]), sep = "")
```

We now plot Figure 3 in the main article.

```
# Open a graphical window. If plotting into a file, skip the next line.
sizeGrWindow(7, 7); fp = FALSE;
#pdf(fi = spaste("Plots/humanChimp-motivationFigure-dendrosAndTable.pdf"), w = 7, h = 7.0); fp = TRUE
layout(matrix(c(1,2,5, 3,4,5), 3, 2),
      heights = c(3, 1, 5.5), widths = c(1, 1));
#layout.show(5);

par(mgp = c(3, 1, 0));
plotDendroAndColors(dendrograms[[1]],
                  cbind(colorHuman[inNetwork], colorChimp[inNetwork]),
                  c("Human modules", "Chimp modules"),
                  setLayout = FALSE,
                  marAll = c(1, 6, 2.7, 0.2),
                  addGuide = FALSE,
                  main = "A. Human gene dendrogram\nand module colors", cex.main = 1.2,
                  dendroLabels = FALSE, hang = 0.03, cex.colorLabels = 0.7, abHeight = 0.95);

par(mgp = c(3, 1, 0));
plotDendroAndColors(dendrograms[[2]],
                  cbind(colorHuman[inNetwork], colorChimp[inNetwork]),
                  c("Human modules", "Chimp modules"),
                  setLayout = FALSE,
                  marAll = c(1, 6, 2.7, 0.2),
                  addGuide = FALSE,
                  main = "B. Chimp gene dendrogram\nand module colors", cex.main = 1.2,
                  dendroLabels = FALSE, hang = 0.03, cex.colorLabels = 0.7, abHeight = 0.95);
# Plot the overlap table
fcex = 1.00;
pcex = 1.0
fcexl = 1.00;
pcexl = 1.00;
par(mar = c(6, 7, 2, 1.0));
labeledHeatmap(Matrix = numMat,
             xLabels = xLabels, xSymbols = xSymbols,
             yLabels = yLabels, ySymbols = ySymbols,
             colorLabels = TRUE,
             colors = greenWhiteRed(100)[50:100],
             textMatrix = textMat, cex.text = if (fp) fcex else pcex, setStdMargins = FALSE,
             cex.lab = if (fp) fcexl else pcexl,
             xColorWidth = 0.08,
             main = "C. Human modules (rows) vs. Chimp modules (columns)", cex.main = 1.2);

dev.off();
```

The second plot (Figure 4 in the main paper) shows a module eigene heatmap and connectivity scatterplots for two human modules, yellow and blue. The yellow module is highly preserved in the chimp samples, while the blue module is preserved less strongly. Before we begin creating the figure, we prepare color-coded indicators of brain areas from which the samples were taken:

```
# Use human set as reference, chimp as test
ref = 1;
test = 2
# The area from which each sample waas taken is encoded in the sample names.
sampleNames = rownames(multiExpr[[ref]]$data)
cortical = substring(sampleNames, 4) %in% c("Brocas.", "prv", "prf", "acc", "Brocas")
cn = substring(sampleNames, 4) %in% c("CN");
cerebellum = substring(sampleNames, 4) %in% c("VC");
# Cortical samples will be coded lightblue
sampleIndicator = c("lightblue", "white")[2-as.numeric(cortical) ];
# CN samples are coded orange
sampleIndicator[cn] = "orange";
# Cerebellum samples are code turquoise
sampleIndicator[cerebellum] = "magenta";
```

We now create Figure 4 in the main paper. As it stands, the code will plot to the screen. Uncommenting the line starting with `pdf(` (and skipping the `sizeGrWindow`) will instead put the plot into a pdf file. If plotting into the file, the directory `Plots` must exist in the current directory (or the user can change the file name to suit).

```
# We will plot these two modules. Modify to get plots for other modules.
plotMods = c("yellow", "blue");
# Open a graphical window. If plotting into a file, skip the next line and instead uncomment and paste the
# line starting with #pdf.
sizeGrWindow(10,5.5);
#pdf(fi = spaste("Plots/humanChimp-motivationFigure-moduleSpecific.pdf"), w = 10, 5.5); fp = TRUE
# Set the plot layout (sectioning). Type help(layout) to see what layout does.
layout(matrix(c(1,2,6,7, 3,3,8,8, 4,4,9,9, 5,5,10,10), 4, 4),
            heights = rep(c(1.0, 1.0), 2));
# Optional: show the sectioning. Skip if plotting into a file.
layout.show(10);
# Keep track of number of plotted panels.
ind = 1;
# Loop over modules to be displayed.
for (mod in plotMods)
{
   # The next big chunk of the code creates a module heatmap and an eigengene plot beneath. Samples are
   # ordered by the area from which they were taken. We promise to one day create a simple function that
   # will create this plot in a single call.
   # Here we prepare various variables for the plot
   order = order(sampleIndicator)
   refX = 1;
   modx = match(mod, substring(colnames(mes[[refX]]), 3));
   if (refX==1) par(mar = c(0.2,2.5,3,0.2)) else par(mar = c(0.2,0.2,3,0.5))
   nModGenes = sum(colorList[[ref]]==mod);
   modExpr = scale(multiExpr[[ref]]$data[, colorList[[ref]]==mod])
   geneOrder = hclust(as.dist(1-cor(modExpr, use="p")), method = "average")$order
   # sampleOrder = hclust(dist(modExpr), method = "average")$order;
   sampleOrder = order;
   MaxExpr = max(modExpr, na.rm = TRUE);
   MinExpr = min(modExpr, na.rm = TRUE);
   MaxZ = max(MaxExpr, abs(MinExpr));
   zlim = c(-MaxZ, MaxZ);
   palette = c("grey", greenWhiteRed(50));
   exprColIndex = (modExpr + MaxZ)/2/MaxZ * 49 + 2;
   exprColIndex[is.na(exprColIndex)] = 1;
```

```
eigengene = mes[[refX]][, modx]
# Set up the plotting coordinates by plotting an empty barplot
par(mgp = c(0, 0, 0));
if (refX==1)
{
   bp = barplot(as.vector(eigengene[sampleOrder]), col = "white", border = "white", axisnames = FALSE,
               main = spaste(LETTERS[ind], ". Module ", mod, " in human"),
               axes = FALSE, ylab = "Module expression\nheatmap", cex.lab = 0.9);
} else {
   bp = barplot(as.vector(eigengene[sampleOrder]), col = "white", border = "white", axisnames = FALSE,
               main = spaste("male CTX95"),
               axes = FALSE, ylab = "", cex.lab = 0.9);
}
ind = ind + 1;

# Get the coordinates of the plot rectangle
plotbox = par("usr");
xstep = bp[2]-bp[1]; xLeft = bp - xstep/2; xRight = bp + xstep/2;
nrows = ncol(modExpr);
yRange = plotbox[4]-plotbox[3];
yBot = plotbox[3] + c(0:(nrows-1)) * yRange/nrows;
yTop = yBot + yRange/nrows;
# Plot the actual heatmap
for (sample in 1:nrow(modExpr))
{
  colorInd = as.integer(exprColIndex[sampleOrder[sample], geneOrder ]);
  rect(xleft = rep(xLeft[sample], nrows), xright = rep(xRight[sample], nrows),
       ybottom = yBot, ytop = yTop, col = palette[colorInd], border = palette[colorInd]);
}

# ----------------------------------------------------------------------------------
# The next chunk of code plots the color-coded eigengene barplot.
if (refX==1) par(mar = c(2.4, 2.5, 0.2, 0.2)) else par(mar = c(2,0.2,0.2,0.5))
par(mgp = c(0.5, 0, 0));
MaxE = max(abs(eigengene), na.rm = TRUE);
eigengene[is.na(eigengene)] = 0;
index = as.integer(as.vector(eigengene + MaxE)/(2*MaxE) * 50 + 1);
Colors = greenWhiteRed(50)[index];
barplot(as.vector(eigengene[sampleOrder]), col = Colors[sampleOrder],
       xlab = "", ylab = if (refX==1) "Eigengene expression" else "", cex.lab = 0.9, yaxt = "none");

# ----------------------------------------------------------------------------------
# This part plots the sample indicator as a color-coded bar.
plotbox = par("usr"); yRange = plotbox[4] - plotbox[3]
rect(xleft = xLeft, xright = xRight, ytop = plotbox[3] - yRange/10, ybot = plotbox[3] - 1.5 *yRange/10,
     col = sampleIndicator[sampleOrder],
     border = sampleIndicator[sampleOrder], xpd = TRUE);
text(mean(c(xLeft, xRight)), plotbox[3] - 2.2*yRange/10, "Sample brain area indicator", adj = c(0.5,1),
     xpd = TRUE, cex = 0.9);

# This concludes the heatmap and eigengene barplots. The scatterplots are much easier to generate.
# Restrict to module genes:
modGenes = colorList[[ref]]==mod;

# ----------------------------------------------------------------------------------
# Calculate intramodular correlations and adjacencies
corRef = vectorizeMatrix(cor(multiExpr[[ref]]$data[, modGenes], use = "p"));
corTest= vectorizeMatrix(cor(multiExpr[[test]]$data[, modGenes], use = "p"));
adjRef = adjacency(multiExpr[[ref]]$data[, modGenes], power = 6, type = "signed hybrid");
```

```
  adjTest= adjacency(multiExpr[[test]]$data[, modGenes], power = 6, type = "signed hybrid");
  # Select a sample to plot (using all gene-gene pairs would result in a needlessly large plot)
  set.seed(10)
  select = sample(1:length(corRef), 3000);
  # Set margin parameters
  mar = c(3.8, 3.8, 3.8, 0.7)
  mgp = c(2.1, 0.7, 0);
  par(mar = mar);
  par(mgp = mgp);
  # Create the scatterplot. Note that the correlation and p-value are calculated from all data, the sample
  # only used for plotting.
  verboseScatterplot(corRef, corTest, sample = select,
                     xlab = spaste("Correlation in human data"),
                     ylab = spaste("Correlation in chimp data"),
                     abline = TRUE,
                     main = spaste(LETTERS[ind], ". Intramodular correlation\n",
                                   "in human ", mod, " module\n"),
                     corLabel = "cor.cor",
                     cex.main = 1.2, cex.lab = 1, cex.axis = 1, cex = 0.6)
  ind = ind + 1;

  # -------------------------------------------------------------------------------------
  # We now plot a scatterplot of intramodular connectivities. We first calculate kIM:
  diag(adjRef) = NA;
  diag(adjTest) = NA;
  kIMRef = apply(adjRef, 1, sum, na.rm = TRUE);
  kIMTest = apply(adjTest, 1, sum, na.rm = TRUE);
  # Plot the scatterplot:
  color = 1;
  verboseScatterplot(kIMRef, kIMTest,
                     xlab = spaste("kIM in human data"),
                     ylab = spaste("kIM in chimp data"),
                     abline = TRUE,
                     main = spaste(LETTERS[ind], ". Intramodular connectivity\n",
                                   "in human ", mod, " module\n"), corLabel = "cor.kIM",
                     cex.main = 1.2, cex.lab = 1, cex.axis = 1, cex = 0.6, col = color)

  ind = ind + 1;

  # -------------------------------------------------------------------------------------
  # Lastly, we plot a scatterplot of module eigengene-based connectivities:
  # Calculate module eigengene-based connectivities
  kMERef = cor(multiExpr[[ref]]$data[, modGenes], mes[[ref]][, modx], use = "p")
  kMETest= cor(multiExpr[[test]]$data[, modGenes], mes[[test]][, modx], use = "p")
  # Plot the scatterplot
  verboseScatterplot(kMERef, kMETest,
                     xlab = spaste("kME in human data"),
                     ylab = spaste("kME in chimp data"),
                     abline = TRUE,
                     main = spaste(LETTERS[ind], ". Eigengene-based connect.\n",
                                   "in human ", mod, " module\n"), corLabel = "cor.kME",
                     cex.main = 1.2, cex.lab = 1, cex.axis = 1, cex = 0.6, col = color)
  ind = ind + 1;
}
# If plotting into a pdf file, close the file. An opened file cannot be used.
dev.off();
```

A Network analysis of expression data

In this section we briefly describe the network construction and module identification methods that were used to construct the network modules whose preservation we study. The human modules were originally constructed in [2] using a weighted gene network with soft thresholding power $\beta = 9$, and a constant-height branch cut to identify gene modules in the hierarchical clustering tree (dendrogram). The original work [2] did not identify modules in the chimp network. Here we recreate the human network and human modules, and we apply the same module detection procedure on the chimp data set to obtain chimp modules.
To run this code, the user needs to execute all code in Sections 1.a and 2.a. Thus, here we assume the R session is already set up, the expression data is loaded, and the multi-set expression variable `multiExpr` has been created.
We start the network analysis by calculating the human and chimp adjacencies of the 4000 genes:

```
powerHuman=9
powerChimp=powerHuman
# Human network:
adjHuman = adjacency(multiExpr$Human$data, power = powerHuman);
# Chimp network:
adjChimp = adjacency(multiExpr$Chimp$data, power = powerChimp);
```

Next, we calculate the connectivities of all genes in the networks and scale them by the corresponding maximum connectivity:

```
ConnectivityHuman = ConnectivityHuman/max(ConnectivityHuman);
ConnectivityChimp = ConnectivityChimp/max(ConnectivityChimp);
```

We now find genes whose scaled connectivity is at least 0.1 in at least one of the networks:

```
minconnections=.1
rest1=ConnectivityChimp>minconnections | ConnectivityHuman>minconnections
table(rest1)
```

A total of 2241 genes pass this threshold. We restrict the adjacencies to the highly connected genes:

```
adjChimpR=adjChimp[rest1,rest1]
adjHumanR=adjHuman[rest1,rest1]
rm(adjChimp);rm(adjHuman); collectGarbage()
```

To obtain modules, we first calulcate the TOM-based dissimilarity [3], then feed the dissimilarity into hierarchical clustering implemented in the function `flashClust`:

```
distTOMChimp = TOMdist(adjChimpR)
distTOMHuman = TOMdist(adjHumanR)
collectGarbage()
# Hierarchical clustering:
hierTOMHuman = flashClust(as.dist(distTOMHuman),method="average")
hierTOMChimp = flashClust(as.dist(distTOMChimp),method="average")
```

Following [2], we identify modules using the constant-height branch cut with cut height 0.95 and minimum module size of 30:

```
colorh1=as.character(cutreeStaticColor(hierTOMHuman,cutHeight=.95,minSize=30))
table(colorh1)
colorh2=as.character(cutreeStaticColor(hierTOMChimp,cutHeight=.95,minSize=30))
table(colorh2)
```

We now plot the human and chimp dendrograms and the corresponding module colors:

```
# Open a suitably sized graphics window
sizeGrWindow(9, 5);
# Alternatively, uncomment the next line to open a pdf file that will hold the plot.
# pdf(file="Plots/HumanChimp-geneDendrogramsAndModuleColors.pdf", wi=9, he=5)
# Set figure layout
layout(matrix(c(1:4), 2,2), heights = c(0.8, 0.2));
# Plot the dendrograms and module colors
plotDendroAndColors(hierTOMHuman,
                  cbind(colorh1, colorh2),
                  c("Human colors", "Chimp colors"),
                  main="Human brain", dendroLabels=FALSE,
                  setLayout = FALSE, abHeight = 0.95,
                  marAll = c(0.3, 4.5, 2, 0.2))
plotDendroAndColors(hierTOMChimp,
                  cbind(colorh1, colorh2),
                  c("Human colors", "Chimp colors"),
                  main="Chimp brain", dendroLabels=FALSE,
                  setLayout = FALSE, abHeight = 0.95)
# If plotting into a file, close it.
dev.off();
```

The dendrograms are the same as those in Figure 3 in the main article. We now create color vectors for all genes in the analysis. The genes that did not pass the minimum connectivity threshold will be assigned the color grey, the color of unassigned genes. At the end we save the module color information; the data is used in Section 2.b.

```
colorh=as.character(colorh1)
colorhALL=rep("grey", length(ConnectivityHuman))
colorhALL[rest1]=as.character(colorh)
colorh=as.character(colorh2)
colorh2ALL=rep("grey", length(ConnectivityChimp))
colorh2ALL[rest1]=as.character(colorh)
# Give the module colors more descriptive names
colorHuman = colorhALL;
colorChimp = colorh2ALL;
# Also save the indicator of genes that pass the connectivity threshold
inNetwork = rest1
# Save module colors (assignments) and the indicator
save(colorHuman, colorChimp, inNetwork,
    file = "HumanChimp-OldhamAnalysis-colorHuman-colorChimp-inNetwork.RData")
```

References

[1] Amy V. Kapp and Robert Tibshirani. Are clusters found in one dataset present in another dataset? *Biostat*, 8(1):9–31, 2007.

[2] MC Oldham, S Horvath, and DH Geschwind. Conservation and evolution of gene coexpression networks in human and chimpanzee brains. *PNAS*, 103(47):17973–17978, 2006.

[3] B Zhang and S. Horvath. General framework for weighted gene coexpression analysis. *Statistical Applications in Genetics and Molecular Biology.*, 4(17), 2005.