Connect modules between stages

```
suppressWarnings(library("knitr"))
suppressWarnings(library("gplots"))
suppressWarnings(library("igraph"))
opts_chunk$set(tidy.opts=list(width.cutoff=80),tidy=TRUE,dev="png",dpi=150)
```

define the functions that will be used for extracting useful data

```
load_obj <- function(file.path) {
    temp.space <- new.env()
    obj <- load(file.path, temp.space)
    obj2 <- get(obj, temp.space)
    rm(temp.space)
    return(obj2)
}

maxScl <- function(df, dir = "row", max_value = NULL, log_space = TRUE) {
    if (dir == "row") {
        dir = 1
    } else if (dir == "col") {
        dir = 2
    } else {
        print("dir must be 'row' or 'col'.")
        return
    }
    if (is.null(max_value)) {
        max_value = median(apply(df, dir, max))
    }
    if (log_space) {
        df = expm1(df)
        max_value = expm1(max_value)
    }
    df_scl = sweep(df, dir, apply(df, dir, max), "/")
    df_scl = df_scl * max_value
    if (log_space) {
        df_scl = log1p(df_scl)
    }
    return(df_scl)
}

module.match <- function(NMF_list, ref_rep = "rep0", min.cor = 0.5, verbose = F) {
    NMF_Ms = list()
    NMF_Gs = list()
    NMF_Gs[[ref_rep]] = maxScl(NMF_list[[ref_rep]][["G"]], dir = "col", max_value = 1,
        log_space = F)
    num_M = dim(NMF_Gs[[ref_rep]])[2]
    num_gen = dim(NMF_Gs[[ref_rep]])[1]
    for (rep in setdiff(names(NMF_list), ref_rep)) {
        ## match modules across different replicates
        n_G = NMF_list[[rep]][["G"]]
        n_G = maxScl(n_G, dir = "col", max_value = 1, log_space = F)
        NMF_Gs[[rep]] = n_G
        match_ind = c()
        for (i in 1:num_M) {
            vec = NMF_Gs[[ref_rep]][, i]
            vec_ind = which(vec > 0.1)
            G_ind = which(apply(n_G, 1, max) > 0.1)
            use_ind = union(vec_ind, G_ind)
            cors = cor(vec[use_ind], n_G[use_ind, ])
            cor_ind = which.max(cors)
            if (max(cors) < min.cor) {
                ## SHOULD RECORD THE CORRELATION SCORES AND SEE IF THERE ARE AMBIGUOUS MATCHES
                match_ind = c(match_ind, NaN)
                # print(paste(stage,'rep0 module',i-1,'has no match in',rep))
                NMF_Gs[[rep]][, i] = NaN
            } else {
                NMF_Gs[[rep]][, i] = n_G[, cor_ind]
                match_ind = c(match_ind, cor_ind)
```

```
            }
        }
        if (verbose) {
            if (num_M != length(unique(match_ind))) {
                print(paste(stage, rep, " modules with duplicated matches:"))
                for (i in match_ind[which(duplicated(match_ind))]) {
                  if (!is.na(i)) {
                    print(paste(toString(which(match_ind == i) - 1), "from rep0 matched to",
                      i - 1))
                  }
                }
                print(paste("modules with no match", toString(setdiff(c(1:num_M),
                  unique(match_ind)) - 1)))
            }
        }
        # NMF_Gs[[paste0('DS',stage)]][[rep]]=maxScl(NMF_Gs[[paste0('DS',stage)]][[rep]],dir
        # = 'col') NMF_tops[[rep]]=NMF_list[[rep]][['top30genes']]
    }
    num_reps = length(names(NMF_list))
    for (m in colnames(NMF_Gs[[ref_rep]])) {
        NMF_Ms[[m]] = matrix(nrow = num_gen, ncol = num_reps)
        rownames(NMF_Ms[[m]]) = rownames(NMF_Gs[[ref_rep]])
        colnames(NMF_Ms[[m]]) = paste0("rep", c(0:(num_reps - 1)))
        for (rep in names(NMF_list)) {
            NMF_Ms[[m]][, rep] = NMF_Gs[[rep]][, m]
        }
        max_w = apply(NMF_Ms[[m]], 1, function(x) max(x, na.rm = T))
        # min_w=apply(NMF_Ms[[paste0('DS',stage)]][[paste0('Module',m)]],1,min)
        NMF_Ms[[m]] = NMF_Ms[[m]][-which(max_w < 0.1), , drop = F]
        re_ind = order(NMF_Ms[[m]][, ref_rep], decreasing = T)
        NMF_Ms[[m]] = NMF_Ms[[m]][re_ind, ]
        # print(paste('Module',m,'# kept
        # genes:',dim(NMF_Ms[[paste0('DS',stage)]][[paste0('Module',m)]])[1]))
    }
    return(NMF_Ms)
}
```

Load the NMF results for each stage (these data files are not provided due to their large sizes)

A best K (number of modules or `n_component` argument used for running NMF) is picked for each stage based on the stability of the results from 10 NMF runs with random initial conditions.

```
ZFHIGH_k = c(10)
ZFOBLONG_k = c(11)
ZFDOME_k = c(17)
ZF30_k = c(15)
ZF50_k = c(20)
ZFS_k = c(25)
ZF60_k = c(25)
ZF75_k = c(24)
ZF90_k = c(45)
ZFB_k = c(40)
ZF3S_k = c(31)
ZF6S_k = c(42)

stages = c("ZFHIGH", "ZFOBLONG", "ZFDOME", "ZF30", "ZF50", "ZFS", "ZF60", "ZF75",
    "ZF90", "ZFB", "ZF3S", "ZF6S")

zf_C <- list()
zf_G <- list()
zf_top <- list()
zf_genes = c()
NMF_list = list()
module_match = list()
for (stage in stages) {
    stage_k = get(paste0(stage, "_k"))[1]
    NMF_obj = load_obj(paste0("~/Dropbox/Desktop_Laptop/Data and analysis/Final scripts/NMF/Results/DS_stages/DS_",
```

```
        stage, "/result_tbls.Robj"))
    NMF_list[[stage]] = NMF_obj[[paste0("K=", stage_k)]][["rep0"]]
    zf_C[[stage]] = data.frame(NMF_list[[stage]][["C"]], stringsAsFactors = F)
    zf_G[[stage]] = data.frame(NMF_list[[stage]][["G"]], stringsAsFactors = F)
    colnames(zf_G[[stage]]) = rownames(zf_C[[stage]])
    zf_genes = c(zf_genes, rownames(zf_G[[stage]]))
    zf_top[[stage]] = data.frame(NMF_list[[stage]][["top30genes"]], stringsAsFactors = F)
    module_match[[stage]] = module.match(NMF_obj[[paste0("K=", stage_k)]], ref_rep = "rep0",
        min.cor = 0.5)
}
```

Find and remove modules that are primarily driven by batch and noise from each stage

Batch modules are found using the `BatchGene` function in *Seurat* package. Noise modules are defined as the ones that are primarily driven by a single gene (the top ranked gene has a weight more than 3 times the weight of the second ranked gene).

```
library("Seurat")
```

```
## Loading required package: ggplot2
## Warning: package 'ggplot2' was built under R version 3.4.4
## Loading required package: cowplot
## Warning: package 'cowplot' was built under R version 3.4.4
##
## Attaching package: 'cowplot'
## The following object is masked from 'package:ggplot2':
##
##     ggsave
## Loading required package: Matrix
## Warning: package 'Matrix' was built under R version 3.4.4
```

```
rmByCell <- function(scData, low = 1) {
    bData = scData > 0
    # sum up each row in the binary matrix for cell numbers
    num.cell = apply(bData, 1, sum)
    rm.ind = which(num.cell <= low)
    scData.f = scData
    # print(paste('removing',length(rm.ind),'genes...'))
    if (length(rm.ind) > 0) {
        scData.f = scData[-rm.ind, ]
    }
    # now there could be cells with no gene detection. remove them
    rmByGenes(scData.f, lmt = 0)
    return(scData.f)
}

rmByGenes <- function(scData, lmt) {
    # first creat a binary matrix for gene detection
    cptr = scData > 0
    # then sum up each column in the binary matrix for gene numbers
    num.cptr = apply(cptr, 2, sum)
    rm.ind = which(num.cptr <= lmt)
    scData.f = scData
    if (length(rm.ind) > 0) {
        # print(paste('removing',length(rm.ind),'cells with fewer than',lmt,'genes...'))
        scData.f = scData[, -rm.ind]
    }
    # now there could be genes with no detection in any cells. remove them
    cptr = scData.f > 0
    num.cell = apply(cptr, 1, sum)
    rm.ind = which(num.cell == 0)
    if (length(rm.ind) > 0) {
        scData.f = scData.f[-rm.ind, ]
    }
    return(scData.f)
}
```

```
filter.modules <- function(zf_C, zf_G, batch.rm = T, batch.field = 1, nois.rat = 3,
    batch.cutoff = 0.75, verbose = T, module_match = NULL) {
    stages = names(zf_C)
    zf_C_use = list()
    zf_G_use = list()
    module_match_use = list()
    for (stage in stages) {
        batch_module = c()
        if (batch.rm) {
            ZF_seurat = new("seurat", raw.data = zf_C[[stage]])
            # original: ZF_seurat=Setup(ZF_seurat,project='ds',min.cells = 2, names.field =
            # 3,names.delim = '_',do.logNormalize = F,is.expr = 0.01,min.genes = 1)
            ZF_seurat = CreateSeuratObject(zf_C[[stage]], project = "ds", min.cells = 2,
                names.field = batch.field, names.delim = "_", is.expr = 0.01, min.genes = 2)  #do.logNormalize = F,
            # original: cut_off=0.73 cut_off=0.72 if(stage %in% c('B')){ cut_off=0.75 }
            batch.frac = unlist(lapply(levels(ZF_seurat@ident), function(x) sum(ZF_seurat@ident ==
                x)/length(ZF_seurat@ident)))
            batch.use = levels(ZF_seurat@ident)[which(batch.frac > 0.1)]
            batch_module = BatchGene(ZF_seurat, idents.use = batch.use, genes.use = rownames(ZF_seurat@data),
                auc.cutoff = batch.cutoff)
            if (verbose) {
                print(paste("Stage:", stage))
                print(paste("number of batches:", length(batch.use)))
                print(batch.frac)
                print("Batch modules:")
                print(batch_module)
            }
        }
        weigh_st = apply(zf_G[[stage]], 2, sort)
        weigh_rat = weigh_st[dim(weigh_st)[1], ]/weigh_st[dim(weigh_st)[1] - 1, ]
        nois = weigh_rat[which(weigh_rat > nois.rat)]
        if (length(nois) > 0 && verbose) {
            print("Noise modules:")
            print(names(nois))
        }
        batch_module = union(batch_module, names(nois))
        zf_C_use[[stage]] <- zf_C[[stage]][setdiff(rownames(zf_C[[stage]]), batch_module),
            ]
        zf_C_use[[stage]] <- maxScl(zf_C_use[[stage]], max_value = 1, log_space = F)
        zf_G_use[[stage]] <- zf_G[[stage]][, setdiff(colnames(zf_G[[stage]]), batch_module)]
        zf_G_use[[stage]] <- rmByCell(zf_G_use[[stage]], low = 0)
        zf_G_use[[stage]] <- maxScl(zf_G_use[[stage]], max_value = 1, dir = "col",
            log_space = F)
        if (!is.null(module_match)) {
            module_match_use[[stage]] <- module_match[[stage]][setdiff(names(module_match[[stage]]),
                paste0("X", batch_module))]
        }
    }
    if (is.null(module_match)) {
        return(list(C = zf_C_use, G = zf_G_use))
    } else {
        return(list(C = zf_C_use, G = zf_G_use, match = module_match_use))
    }
}
zf_use = filter.modules(zf_C, zf_G, module_match = module_match, batch.cutoff = 0.715,
    batch.field = 3, verbose = F)
```

Print out the size of matrix G at each stage (we will use these matrices to build the tree of connected modules)

```
for (stage in stages) {
    print(stage)
    print(dim(zf_use[["G"]][[stage]]))
}
```

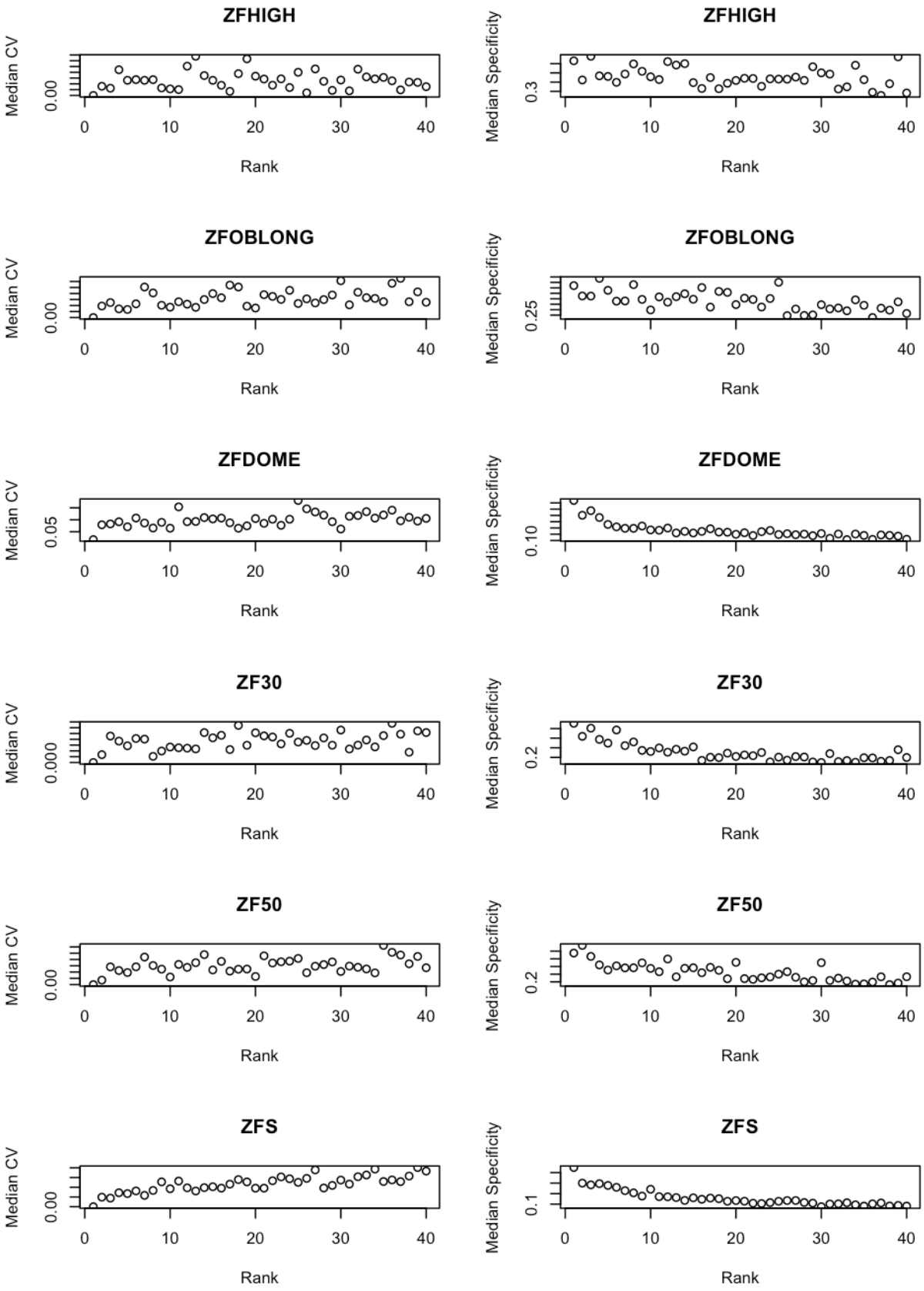
```
## [1] "ZFHIGH"
## [1] 1262    6
## [1] "ZFOBLONG"
## [1] 1166    9
```

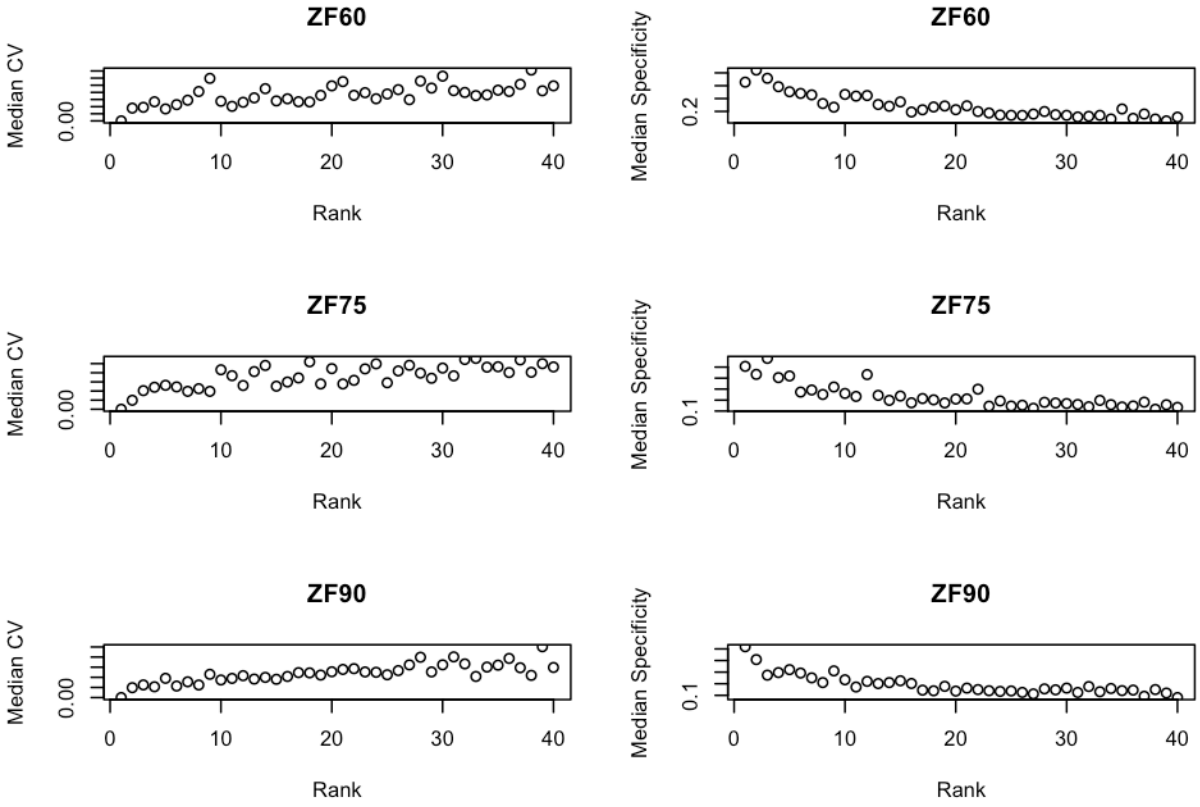
```
## [1] "ZFDOME"
## [1] 1583   17
## [1] "ZF30"
## [1] 1553   13
## [1] "ZF50"
## [1] 1721   13
## [1] "ZFS"
## [1] 1573   25
## [1] "ZF60"
## [1] 1749   20
## [1] "ZF75"
## [1] 1809   19
## [1] "ZF90"
## [1] 1833   28
## [1] "ZFB"
## [1] 1856   27
## [1] "ZF3S"
## [1] 1825   29
## [1] "ZF6S"
## [1] 1854   30
```

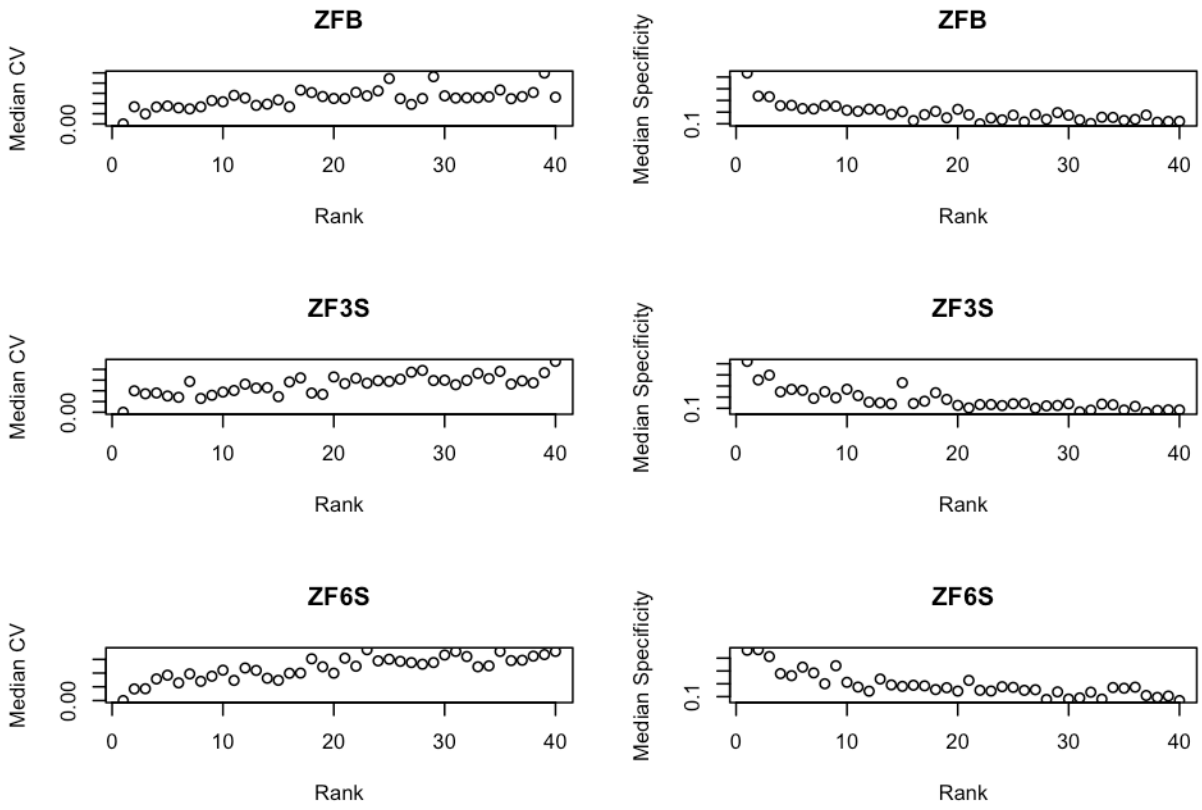
Figure out how many top genes to use for calculating overlap scores

For each stage, calculate the coefficient of variance (CV) and a 'specificity score' for each gene's weight in each module across the 10 replications. To calculate the CV of a gene in a particular module, we first extract that module from each of the 10 NMF runs (modules are matched by correlation of gene weigths). Then the CV of a gene in this module is defined as the mean of its weights divided by the standard deviation of its weights across these 10 replications. The specificity score of a gene for a module is calculated as the weight of the gene in that module divided by the sum of the gene's weights in all modules from the same run. If a gene is a 'robust' marker of the module, it should have a small CV and a high specificity. We then plot these statistics of the top ranking genes in each module to see when CV becomes high and specificity becomes low, indicating the genes ranked higher than that are good genes to use in the overlap score calculations.

```
weight_cv = list()
for (stage in names(zf_use$match)) {
    # num_gen=min(unlist(lapply(zf_use$match[[stage]], function(x) dim(x)[1])))
    num_gen = 40
    weight_cv[[stage]] = matrix(nrow = num_gen, ncol = length(zf_use$match[[stage]]))
    colnames(weight_cv[[stage]]) = names(zf_use$match[[stage]])
    for (m in names(zf_use$match[[stage]])) {
        num_val = min(num_gen, dim(zf_use$match[[stage]][[m]])[1])
        weight_cv[[stage]][1:num_val, m] = apply(zf_use$match[[stage]][[m]][1:num_val,
            ], 1, function(x) sqrt(var(x, na.rm = T))/mean(x, na.rm = T))
    }
}

weight_spec = list()
for (stage in names(zf_use$match)) {
    weight_spec[[stage]] = zf_use$G[[stage]]
    rownames(weight_spec[[stage]]) = 1:dim(zf_use$G[[stage]])[1]
    weight_spec[[stage]] = sweep(weight_spec[[stage]], 1, apply(zf_use$G[[stage]],
        1, sum), "/")
    for (i in 1:dim(zf_use$G[[stage]])[2]) {
        ind_i = order(zf_use$G[[stage]][, i], decreasing = T)
        weight_spec[[stage]][, i] = weight_spec[[stage]][ind_i, i]
    }
}
par(mfrow = c(3, 2))
for (stage in names(weight_cv)) {
    # plot(1:dim(weight_cv[[stage]])[1],apply(weight_cv[[stage]],1,function(x)
    # mean(x,na.rm=T)),main=stage, ylab='Mean CV',xlab='Rank')
    # plot(1:40,apply(weight_spec[[stage]][1:40,],1,function(x)
    # mean(x,na.rm=T)),main=stage, ylab='Mean Specificity',xlab='Rank')
    plot(1:dim(weight_cv[[stage]])[1], apply(weight_cv[[stage]], 1, function(x) median(x,
        na.rm = T)), main = stage, ylab = "Median CV", xlab = "Rank")
    plot(1:40, apply(weight_spec[[stage]][1:40, ], 1, function(x) median(x, na.rm = T)),
        main = stage, ylab = "Median Specificity", xlab = "Rank")
}
```


ZFHIGH
Median CV
0.00
0
10
20
30
40
Rank
ZFHIGH
Median Specificity
0.3
0
10
20
30
40
Rank
ZFOBLONG
Median CV
0.00
0
10
20
30
40
Rank
ZFOBLONG
Median Specificity
0.25
0
10
20
30
40
Rank
ZFDOME
Median CV
0.05
0
10
20
30
40
Rank
ZFDOME
Median Specificity
0.10
0
10
20
30
40
Rank
ZF30
Median CV
0.000
0
10
20
30
40
Rank
ZF30
Median Specificity
0.2
0
10
20
30
40
Rank
ZF50
Median CV
0.00
0
10
20
30
40
Rank
ZF50
Median Specificity
0.2
0
10
20
30
40
Rank
ZFS
Median CV
0.00
0
10
20
30
40
Rank
ZFS
Median Specificity
0.1
0
10
20
30
40
Rank


ZF60
Median CV
0.00
0
10
20
30
40
Rank
ZF60
Median Specificity
0.2
0
10
20
30
40
Rank
ZF75
Median CV
0.00
0
10
20
30
40
Rank
ZF75
Median Specificity
0.1
0
10
20
30
40
Rank
ZF90
Median CV
0.00
0
10
20
30
40
Rank
ZF90
Median Specificity
0.1
0
10
20
30
40
Rank

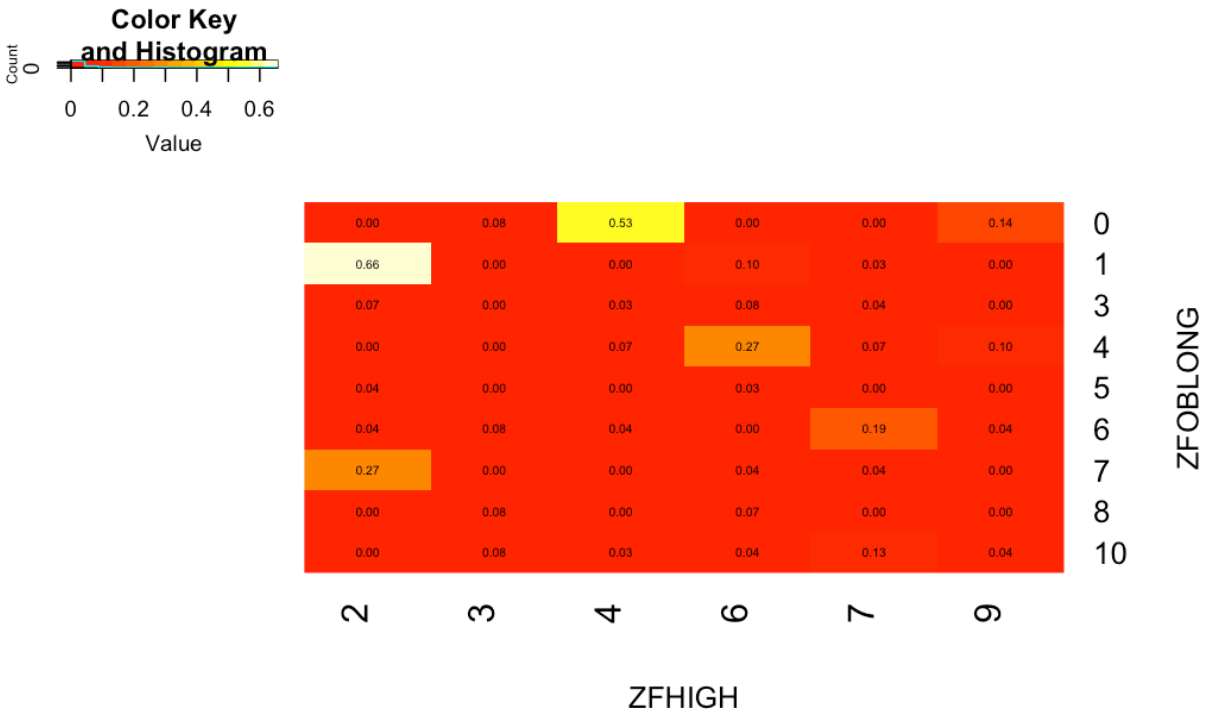


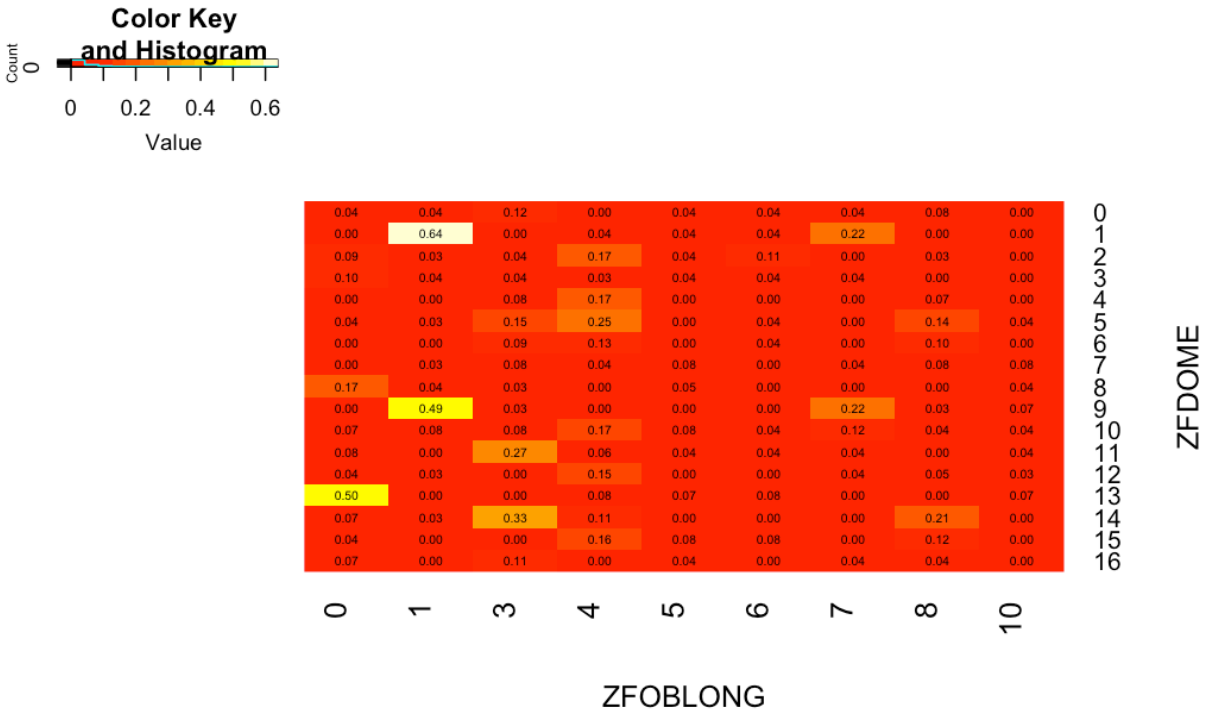
Calculate the weighted overlap between pairs of gene modules in adjacent stages

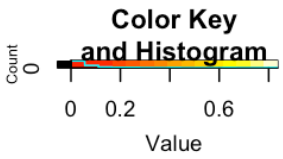
We decided to use the top 25 genes in each module in this calculation. The overlap of two modules is calculated as the sum of the weights of shared genes divided by the sum of weights of all genes. The results of the overlap scores are visualized in heat maps.

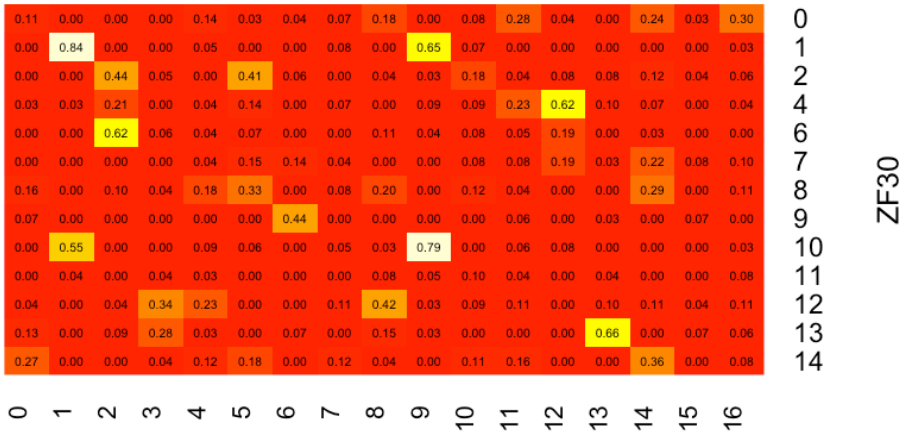
```
Weigh_intersect <- function(M.ind, Data1, Data2, numGene, norm.sum = F) {
    i = M.ind[1]
    j = M.ind[2]
    Data1M = Data1[, i, drop = F]
    Data2M = Data2[, j, drop = F]
    topGenes1 = rownames(Data1)[order(Data1M, decreasing = T)[1:numGene]]
    topGenes2 = rownames(Data2)[order(Data2M, decreasing = T)[1:numGene]]
    if (norm.sum) {
        Data1M = Data1M/sum(Data1M)
        Data2M = Data2M/sum(Data2M)
    }
    inter_genes = intersect(topGenes1, topGenes2)
    weighted_inter = (sum(Data1M[inter_genes, ]) + sum(Data2M[inter_genes, ]))/(sum(Data1M[topGenes1,
        ]) + sum(Data2M[topGenes2, ]))
    return(weighted_inter)
}
Calc_intersect <- function(Data1, Data2, num_top = 25, weigh = F, norm.sum = F) {
    Data1 = sweep(Data1, 2, apply(Data1, 2, max), "/")
    Data2 = sweep(Data2, 2, apply(Data2, 2, max), "/")

    num.spl1 = dim(Data1)[2]
    num.spl2 = dim(Data2)[2]
    cor.M = matrix(0, nrow = num.spl2, ncol = num.spl1)
    num.ind = num.spl1 * num.spl2
    M.ind = vector("list", length = num.ind)
```

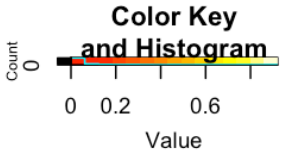
```
    k = 1
    for (i in 1:num.spl1) {
        for (j in 1:num.spl2) {
            M.ind[[k]] = c(i, j)
            k = k + 1
        }
    }

    if (weigh) {
        cor.M.vec = lapply(1:num.ind, function(x) Weigh_intersect(M.ind[[x]], Data1,
            Data2, num_top, norm.sum = norm.sum))
    } else {
        cor.M.vec = lapply(1:num.ind, function(x) length(intersect(rownames(Data1)[order(Data1[,
            M.ind[[x]][1]], decreasing = T)[1:num_top]], rownames(Data2)[order(Data2[,
            M.ind[[x]][2]], decreasing = T)[1:num_top]]))/num_top)
    }

    for (i in 1:num.ind) {
        ind1 = M.ind[[i]][1]
        ind2 = M.ind[[i]][2]
        cor.M[ind2, ind1] = unlist(cor.M.vec[i])
    }
    corDF = data.frame(cor.M, row.names = colnames(Data2))
    colnames(corDF) = colnames(Data1)
    return(corDF)
}

module.overlap <- function(DS_G_use, num.top = 25, weigh = T, heatmap = T, norm.sum = F) {
    stages = names(DS_G_use)
    G_int <- list()
    for (i in 1:(length(stages) - 1)) {
        stage = stages[i]
        stage_next = stages[i + 1]
        # gene_use=intersect(rownames(DS_G_use[[stage]]),rownames(DS_G_use[[stage_next]]))
        G_stage = DS_G_use[[stage]]
        # G_stage=DS_G_use[[stage]][gene_use,]
        G_stage_next = DS_G_use[[stage_next]]
        # G_stage_next=DS_G_use[[stage_next]][gene_use,]
        num_module = dim(G_stage)[2]
        num_module_next = dim(G_stage_next)[[2]]
        G_int[[stage]] <- Calc_intersect(G_stage, G_stage_next, num_top = num.top,
            weigh = weigh, norm.sum = norm.sum)
        ## returns overlap scores in a matrix, colnames are modules at this stage,
        ## rownames are modules at next stage
        if (heatmap) {
            xval <- formatC(as.matrix(G_int[[stage]]), format = "f", digits = 2)
            heatmap.2(as.matrix(G_int[[stage]]), Rowv = FALSE, Colv = FALSE, dendrogram = "none",
                xlab = stage, ylab = stage_next, trace = "none", cellnote = xval,
                notecol = "black", notecex = 0.5)
        }
    }
    return(G_int)
}

G_int = module.overlap(zf_use[["G"]], num.top = 25, weigh = T, heatmap = T, norm.sum = F)
```

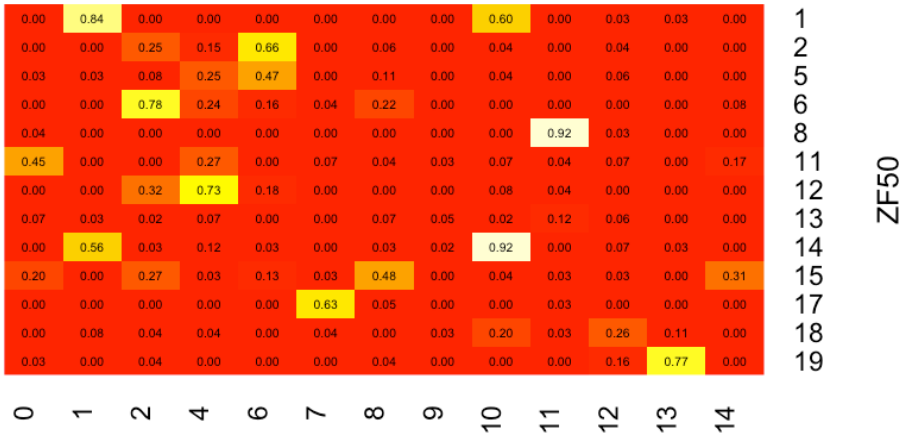

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value
0.00 0.08 0.53 0.00 0.00 0.14
0.66 0.00 0.00 0.10 0.03 0.00
0.07 0.00 0.03 0.08 0.04 0.00
0.00 0.00 0.07 0.27 0.07 0.10
0.04 0.00 0.00 0.03 0.00 0.00
0.04 0.08 0.04 0.00 0.19 0.04
0.27 0.00 0.00 0.04 0.04 0.00
0.00 0.08 0.00 0.07 0.00 0.00
0.00 0.08 0.03 0.04 0.13 0.04
0
1
3
4
5
6
7
8
10
ZFOBLONG
2
3
4
6
7
9
ZFHIGH

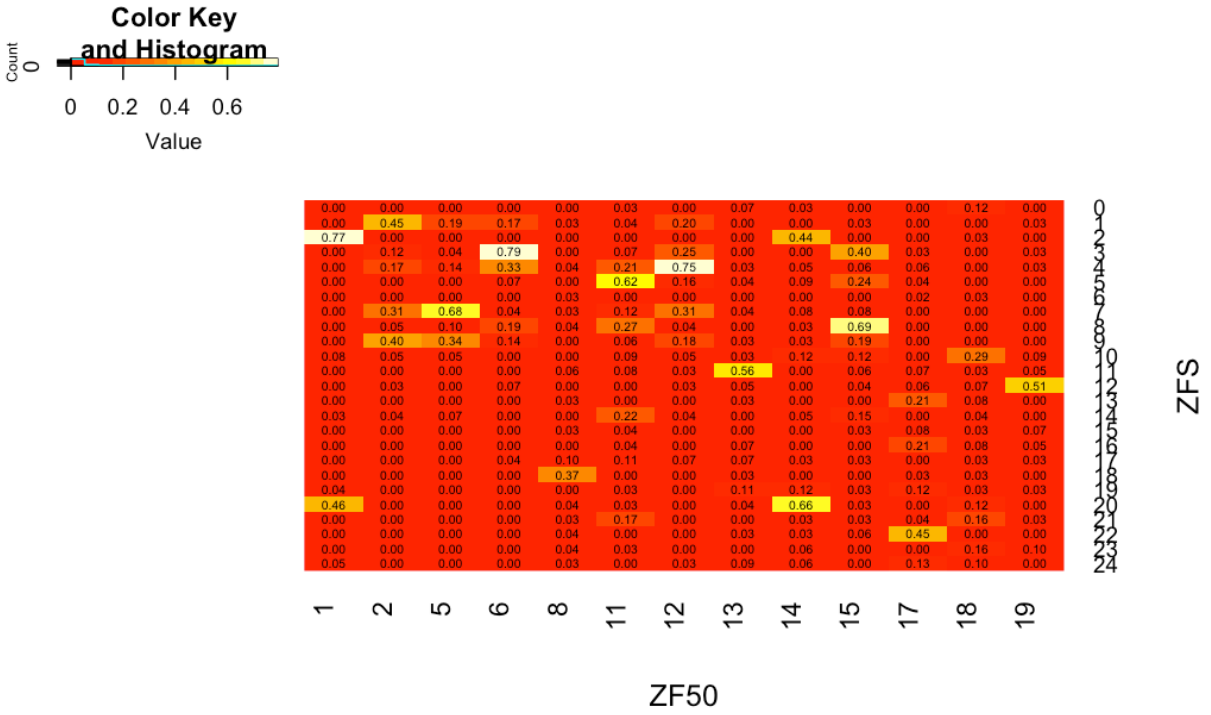

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value
0.04 0.04 0.12 0.00 0.04 0.04 0.04 0.08 0.00
0.00 0.64 0.00 0.04 0.04 0.04 0.22 0.00 0.00
0.09 0.03 0.04 0.17 0.04 0.11 0.00 0.03 0.00
0.10 0.04 0.04 0.03 0.04 0.04 0.04 0.00 0.00
0.00 0.00 0.08 0.17 0.00 0.00 0.00 0.07 0.00
0.04 0.03 0.15 0.25 0.00 0.04 0.00 0.14 0.04
0.00 0.00 0.09 0.13 0.00 0.04 0.00 0.10 0.00
0.00 0.03 0.08 0.04 0.08 0.00 0.04 0.08 0.08
0.17 0.04 0.03 0.00 0.05 0.00 0.00 0.00 0.04
0.00 0.49 0.03 0.00 0.00 0.00 0.22 0.03 0.07
0.07 0.08 0.08 0.17 0.08 0.04 0.12 0.04 0.04
0.08 0.00 0.27 0.06 0.04 0.04 0.04 0.00 0.04
0.04 0.03 0.00 0.15 0.00 0.00 0.04 0.05 0.03
0.50 0.00 0.00 0.08 0.07 0.08 0.00 0.00 0.07
0.07 0.03 0.33 0.11 0.00 0.00 0.00 0.21 0.00
0.04 0.00 0.00 0.16 0.08 0.08 0.00 0.12 0.00
0.07 0.00 0.11 0.00 0.04 0.00 0.04 0.04 0.00
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
ZFDOME
0
1
3
4
5
6
7
8
10
ZFOBLONG

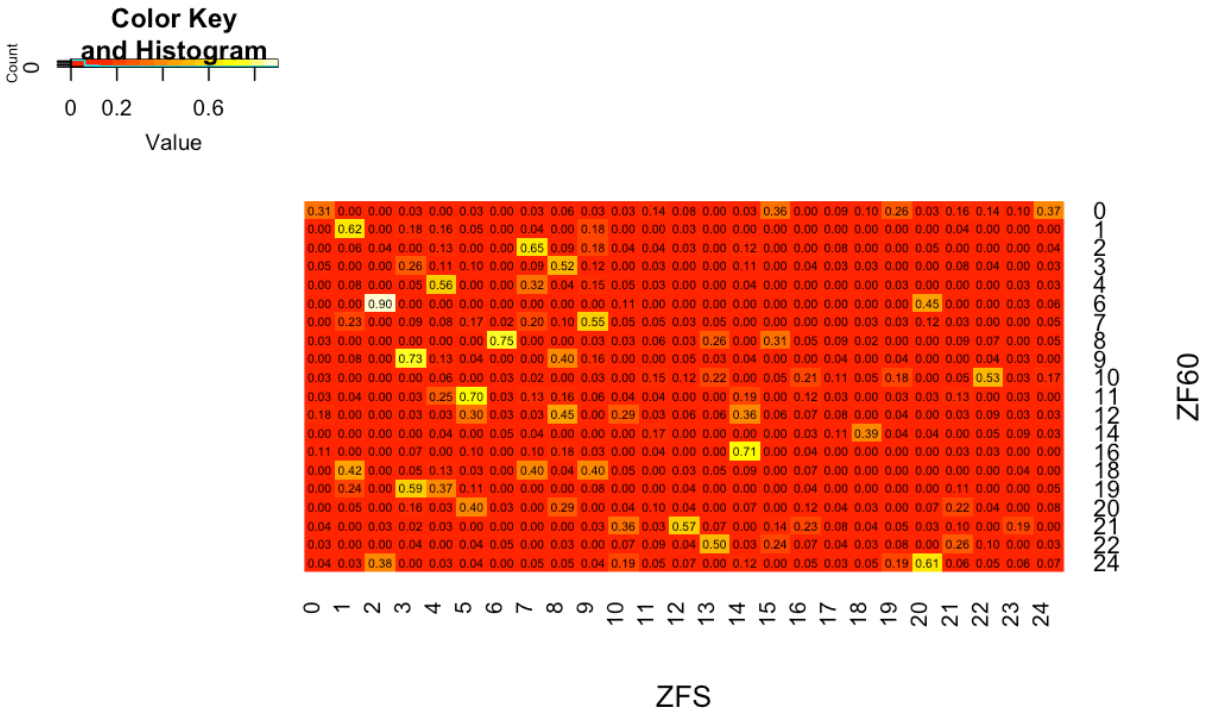

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

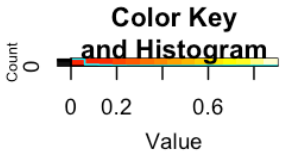

0.11 0.00 0.00 0.00 0.14 0.03 0.04 0.07 0.18 0.00 0.08 0.28 0.04 0.00 0.24 0.03 0.30
0.00 0.84 0.00 0.00 0.05 0.00 0.00 0.08 0.00 0.65 0.07 0.00 0.00 0.00 0.00 0.00 0.03
0.00 0.00 0.44 0.05 0.00 0.41 0.06 0.00 0.04 0.03 0.18 0.04 0.08 0.08 0.12 0.04 0.06
0.03 0.03 0.21 0.00 0.04 0.14 0.00 0.07 0.00 0.09 0.09 0.23 0.62 0.10 0.07 0.00 0.04
0.00 0.00 0.62 0.06 0.04 0.07 0.00 0.00 0.11 0.04 0.08 0.05 0.19 0.00 0.03 0.00 0.00
0.00 0.00 0.00 0.00 0.04 0.15 0.14 0.04 0.00 0.00 0.08 0.08 0.19 0.03 0.22 0.08 0.10
0.16 0.00 0.10 0.04 0.18 0.33 0.00 0.08 0.20 0.00 0.12 0.04 0.00 0.00 0.29 0.00 0.11
0.07 0.00 0.00 0.00 0.00 0.00 0.44 0.00 0.00 0.00 0.00 0.06 0.00 0.03 0.00 0.07 0.00
0.00 0.55 0.00 0.00 0.09 0.06 0.00 0.05 0.03 0.79 0.00 0.06 0.08 0.00 0.00 0.00 0.03
0.00 0.04 0.00 0.04 0.03 0.00 0.00 0.00 0.08 0.05 0.10 0.04 0.00 0.04 0.00 0.00 0.08
0.04 0.00 0.04 0.34 0.23 0.00 0.00 0.11 0.42 0.03 0.09 0.11 0.00 0.10 0.11 0.04 0.11
0.13 0.00 0.09 0.28 0.03 0.00 0.07 0.00 0.15 0.03 0.00 0.00 0.00 0.66 0.00 0.07 0.06
0.27 0.00 0.00 0.04 0.12 0.18 0.00 0.12 0.04 0.00 0.11 0.16 0.00 0.00 0.36 0.00 0.08
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
ZF30
ZFDOME

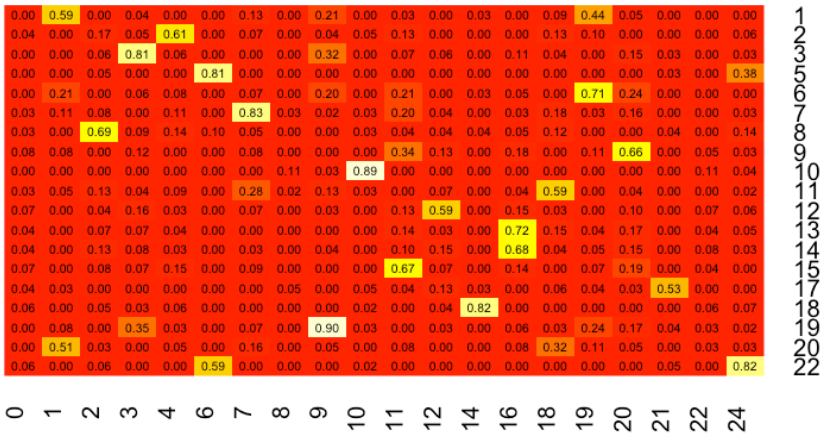

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

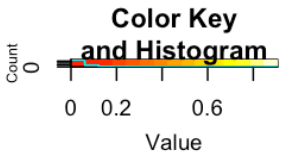

0.00 0.84 0.00 0.00 0.00 0.00 0.00 0.00 0.60 0.00 0.03 0.03 0.00
0.00 0.00 0.25 0.15 0.66 0.00 0.06 0.00 0.04 0.00 0.04 0.00 0.00
0.03 0.03 0.08 0.25 0.47 0.00 0.11 0.00 0.04 0.00 0.06 0.00 0.00
0.00 0.00 0.78 0.24 0.16 0.04 0.22 0.00 0.00 0.00 0.00 0.00 0.08
0.04 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.92 0.03 0.00 0.00
0.45 0.00 0.00 0.27 0.00 0.07 0.04 0.03 0.07 0.04 0.07 0.00 0.17
0.00 0.00 0.32 0.73 0.18 0.00 0.00 0.00 0.08 0.04 0.00 0.00 0.00
0.07 0.03 0.02 0.07 0.00 0.00 0.07 0.05 0.02 0.12 0.06 0.00 0.00
0.00 0.56 0.03 0.12 0.03 0.00 0.03 0.02 0.92 0.00 0.07 0.03 0.00
0.20 0.00 0.27 0.03 0.13 0.03 0.48 0.00 0.04 0.03 0.03 0.00 0.31
0.00 0.00 0.00 0.00 0.00 0.63 0.05 0.00 0.00 0.03 0.00 0.00 0.00
0.00 0.08 0.04 0.04 0.00 0.04 0.00 0.03 0.20 0.03 0.26 0.11 0.00
0.03 0.00 0.04 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.16 0.77 0.00
0 1 2 4 6 7 8 9 10 11 12 13 14
ZF50
ZF30

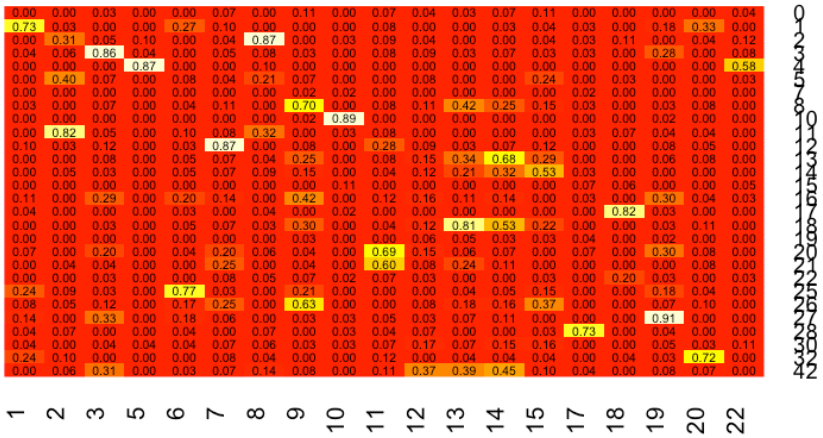

Color Key
and Histogram
Count
0
0 0.2 0.4 0.6
Value
ZFS
ZF50
1 2 5 6 8 11 12 13 14 15 17 18 19
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

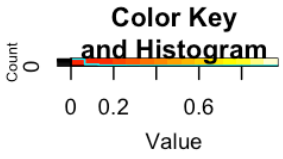

Color Key
and Histogram
Count
0
0 0.2 0.6
Value
ZF60
ZFS
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

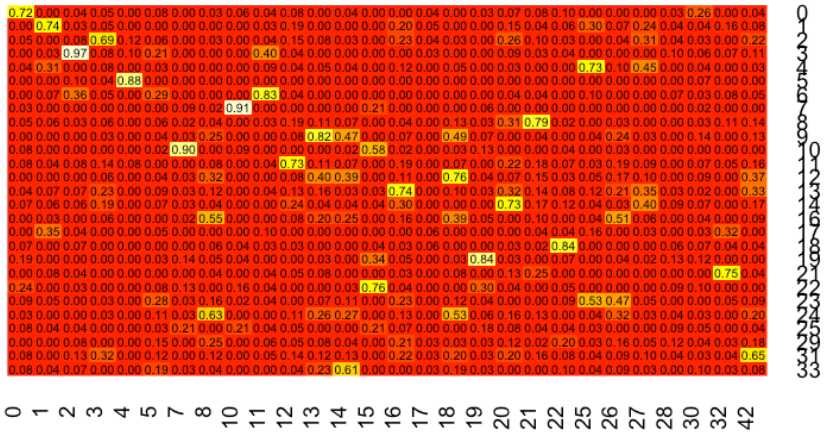

Color Key
and Histogram
Count
0
0
0.2
0.6
Value


ZF75
ZF60

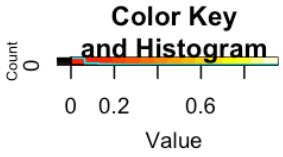

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

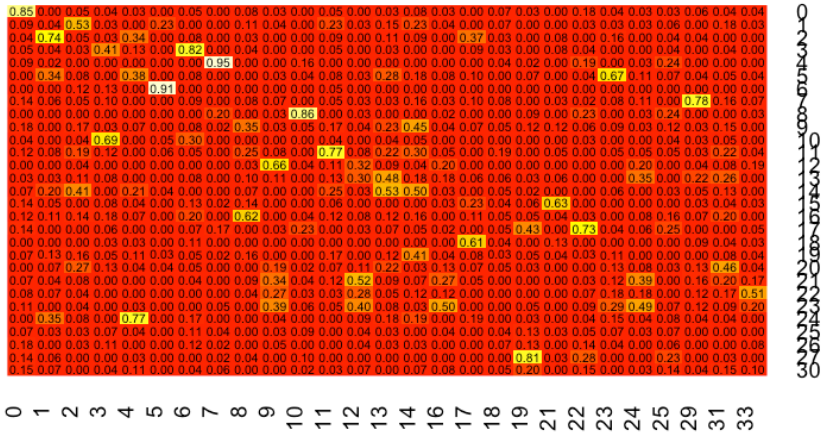

ZF90
ZF75


Color Key
and Histogram
Count
0
0
0.2
0.6
Value

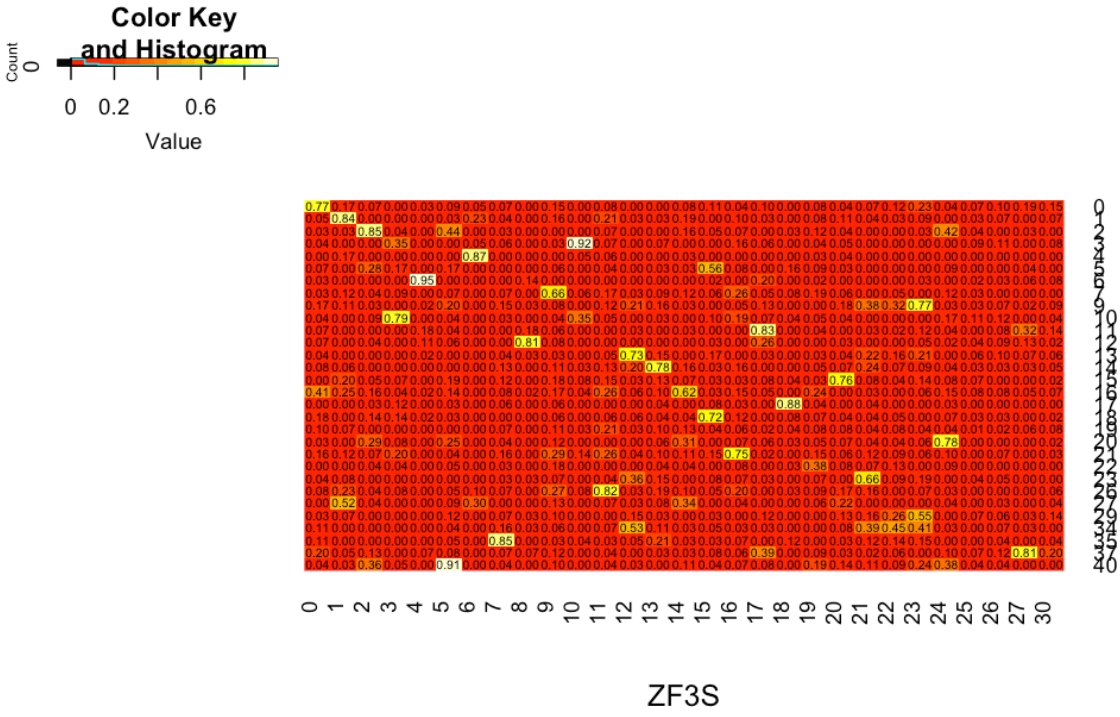

ZFB

ZF90


Color Key
and Histogram
Count
0
0
0.2
0.6
Value


ZF3S

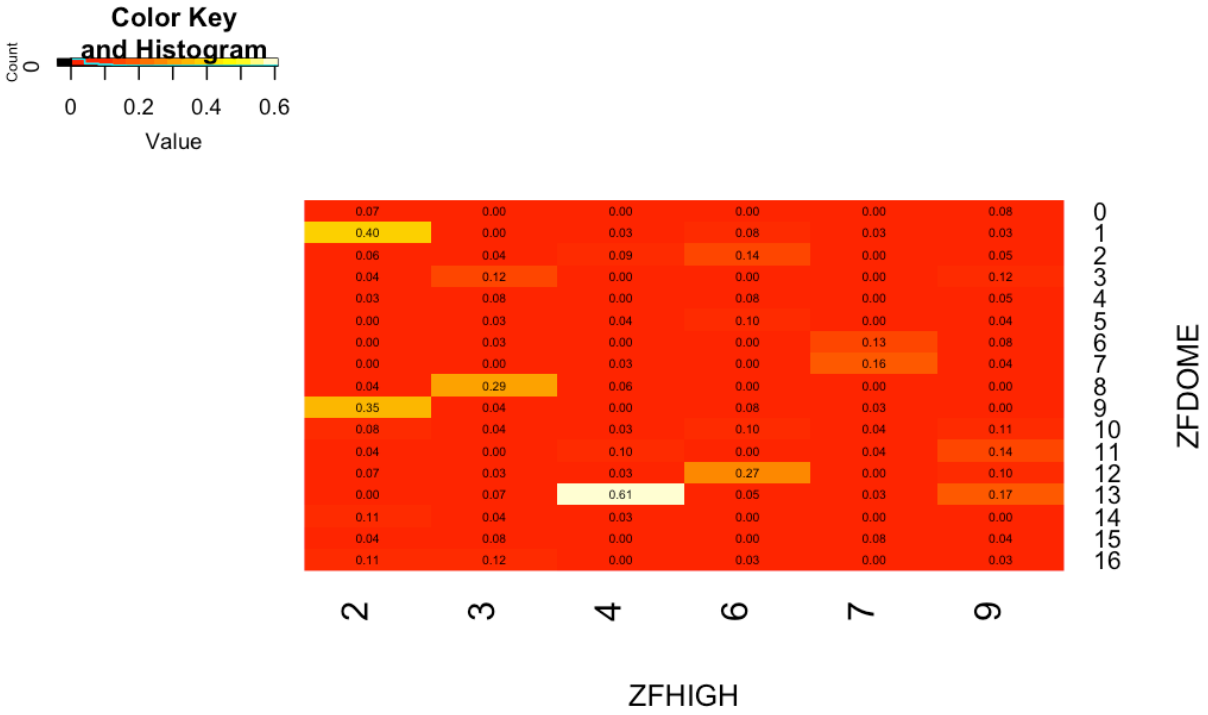
ZFB

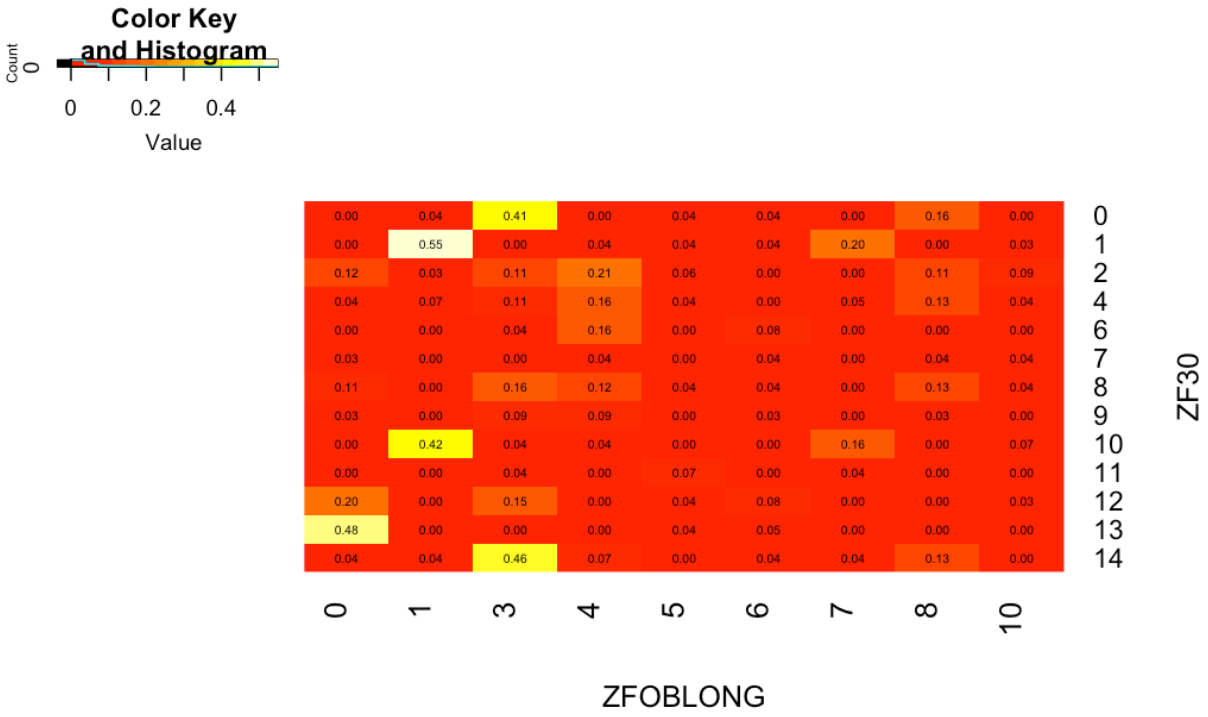


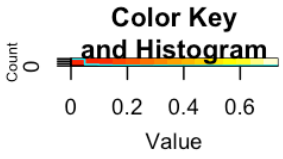
Calculate overlap between modules in every other stage

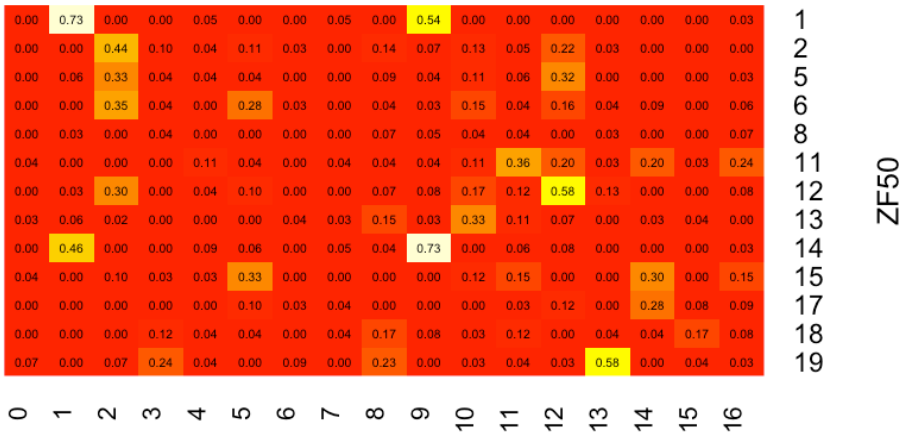
If a stage was not deeply or comprehensively sampled and sequenced, we might not be able to recover certain modules from that stage. This could potentially create dis-connections in the module lineages. In order to produce continuous module lineages when there is potential occasional drop-out of modules, we allow modules separated by one stage to connect to each other when connection to immediate neighbouring stage is not found.

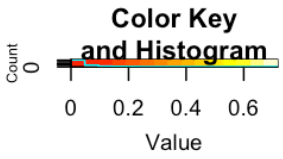
```
module.overlap2 <- function(DS_G_use, num.top = 25, weigh = T, heatmap = T, norm.sum = F) {
    G_int2 <- list()
    stages = names(DS_G_use)
    for (i in 1:(length(stages) - 2)) {
        stage = stages[i]
        stage_next = stages[i + 2]
        gene_use = intersect(rownames(DS_G_use[[stage]]), rownames(DS_G_use[[stage_next]]))
        G_stage = DS_G_use[[stage]][gene_use, ]
        G_stage_next = DS_G_use[[stage_next]][gene_use, ]
        num_module = dim(G_stage)[2]
        num_module_next = dim(G_stage_next)[[2]]
        G_int2[[stage]] <- Calc_intersect(G_stage, G_stage_next, num_top = num.top,
            weigh = weigh, norm.sum = norm.sum)
        ## returns matrix of overlap scores, colnames are modules at this stage, rownames
        ## are modules at next stage
        if (heatmap) {
            xval <- formatC(as.matrix(G_int2[[stage]]), format = "f", digits = 2)
            heatmap.2(as.matrix(G_int2[[stage]]), Rowv = FALSE, Colv = FALSE, dendrogram = "none",
                xlab = stage, ylab = stage_next, trace = "none", cellnote = xval,
                notecol = "black", notecex = 0.5)
        }
    }
    return(G_int2)
}
G_int2 = module.overlap2(zf_use[["G"]], num.top = 25, weigh = T, heatmap = T, norm.sum = F)
```

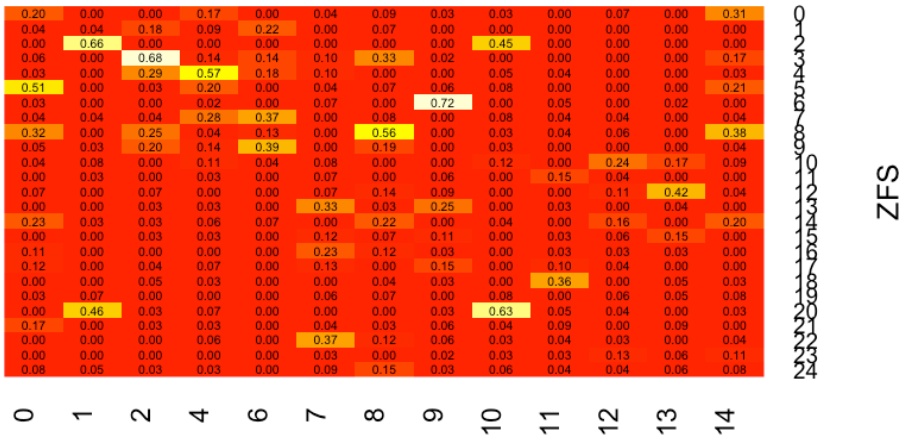

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value
0.07 0.00 0.00 0.00 0.00 0.08
0.40 0.00 0.03 0.08 0.03 0.03
0.06 0.04 0.09 0.14 0.00 0.05
0.04 0.12 0.00 0.00 0.00 0.12
0.03 0.08 0.00 0.08 0.00 0.05
0.00 0.03 0.04 0.10 0.00 0.04
0.00 0.03 0.00 0.00 0.13 0.08
0.00 0.00 0.03 0.00 0.16 0.04
0.04 0.29 0.06 0.00 0.00 0.00
0.35 0.04 0.00 0.08 0.03 0.00
0.08 0.04 0.03 0.10 0.04 0.11
0.04 0.00 0.10 0.00 0.04 0.14
0.07 0.03 0.03 0.27 0.00 0.10
0.00 0.07 0.61 0.05 0.03 0.17
0.11 0.04 0.03 0.00 0.00 0.00
0.04 0.08 0.00 0.00 0.08 0.04
0.11 0.12 0.00 0.03 0.00 0.03
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
ZFDOME
2
3
4
6
7
9
ZFHIGH

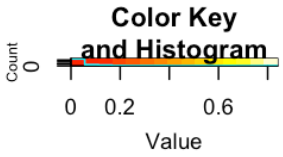

Color Key
and Histogram
Count
0
0
0.2
0.4
Value
0.00 0.04 0.41 0.00 0.04 0.04 0.00 0.16 0.00
0.00 0.55 0.00 0.04 0.04 0.04 0.20 0.00 0.03
0.12 0.03 0.11 0.21 0.06 0.00 0.00 0.11 0.09
0.04 0.07 0.11 0.16 0.04 0.00 0.05 0.13 0.04
0.00 0.00 0.04 0.16 0.00 0.08 0.00 0.00 0.00
0.03 0.00 0.00 0.04 0.00 0.04 0.00 0.04 0.04
0.11 0.00 0.16 0.12 0.04 0.04 0.00 0.13 0.04
0.03 0.00 0.09 0.09 0.00 0.03 0.00 0.03 0.00
0.00 0.42 0.04 0.04 0.00 0.00 0.16 0.00 0.07
0.00 0.00 0.04 0.00 0.07 0.00 0.04 0.00 0.00
0.20 0.00 0.15 0.00 0.04 0.08 0.00 0.00 0.03
0.48 0.00 0.00 0.00 0.04 0.05 0.00 0.00 0.00
0.04 0.04 0.46 0.07 0.00 0.04 0.04 0.13 0.00
0
1
2
4
6
7
8
9
10
11
12
13
14
ZF30
0
1
3
4
5
6
7
8
10
ZFOBLONG

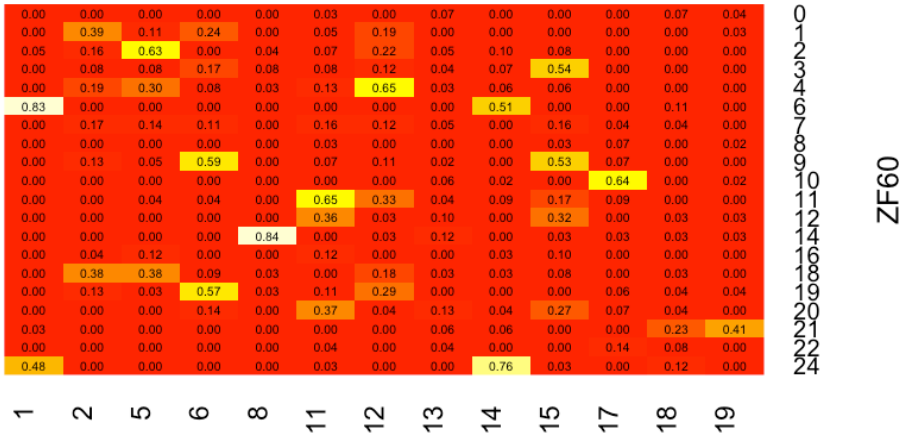

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value

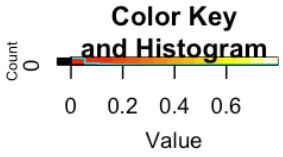

0.00 0.73 0.00 0.00 0.05 0.00 0.00 0.05 0.00 0.54 0.00 0.00 0.00 0.00 0.00 0.00 0.03
0.00 0.00 0.44 0.10 0.04 0.11 0.03 0.00 0.14 0.07 0.13 0.05 0.22 0.03 0.00 0.00 0.00
0.00 0.06 0.33 0.04 0.04 0.04 0.00 0.00 0.09 0.04 0.11 0.06 0.32 0.00 0.00 0.00 0.03
0.00 0.00 0.35 0.04 0.00 0.28 0.03 0.00 0.04 0.03 0.15 0.04 0.16 0.04 0.09 0.00 0.06
0.00 0.03 0.00 0.04 0.00 0.00 0.00 0.00 0.07 0.05 0.04 0.04 0.00 0.03 0.00 0.00 0.07
0.04 0.00 0.00 0.00 0.11 0.04 0.00 0.04 0.04 0.04 0.11 0.36 0.20 0.03 0.20 0.03 0.24
0.00 0.03 0.30 0.00 0.04 0.10 0.00 0.00 0.07 0.08 0.17 0.12 0.58 0.13 0.00 0.00 0.08
0.03 0.06 0.02 0.00 0.00 0.00 0.04 0.03 0.15 0.03 0.33 0.11 0.07 0.00 0.03 0.04 0.00
0.00 0.46 0.00 0.00 0.09 0.06 0.00 0.05 0.04 0.73 0.00 0.06 0.08 0.00 0.00 0.00 0.03
0.04 0.00 0.10 0.03 0.03 0.33 0.00 0.00 0.00 0.00 0.12 0.15 0.00 0.00 0.30 0.00 0.15
0.00 0.00 0.00 0.00 0.00 0.10 0.03 0.04 0.00 0.00 0.00 0.03 0.12 0.00 0.28 0.08 0.09
0.00 0.00 0.00 0.12 0.04 0.04 0.00 0.04 0.17 0.08 0.03 0.12 0.00 0.04 0.04 0.17 0.08
0.07 0.00 0.07 0.24 0.04 0.00 0.09 0.00 0.23 0.00 0.03 0.04 0.03 0.58 0.00 0.04 0.03
1
2
5
6
8
11
12
13
14
15
17
18
19
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
ZF50
ZFDOME

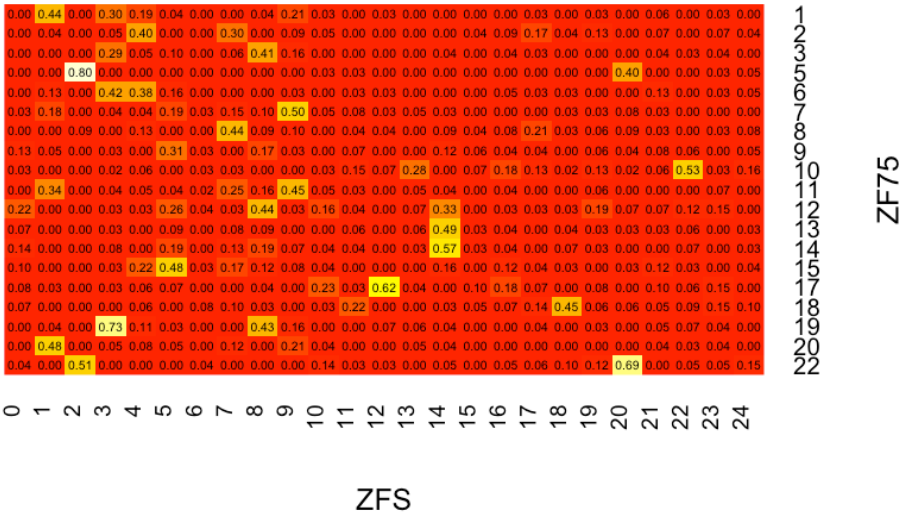

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value

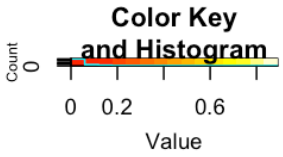

0.20 0.00 0.00 0.17 0.00 0.04 0.09 0.03 0.03 0.00 0.07 0.00 0.31
0.04 0.04 0.18 0.09 0.22 0.00 0.07 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.66 0.00 0.00 0.00 0.00 0.00 0.00 0.45 0.00 0.00 0.00 0.00
0.06 0.00 0.68 0.14 0.14 0.10 0.33 0.02 0.00 0.00 0.00 0.00 0.17
0.03 0.00 0.29 0.57 0.18 0.10 0.00 0.00 0.05 0.04 0.00 0.00 0.03
0.51 0.00 0.03 0.20 0.00 0.04 0.07 0.06 0.08 0.00 0.00 0.00 0.21
0.03 0.00 0.00 0.02 0.00 0.07 0.00 0.72 0.00 0.05 0.00 0.02 0.00
0.04 0.04 0.04 0.28 0.37 0.00 0.08 0.00 0.08 0.04 0.04 0.00 0.04
0.32 0.00 0.25 0.04 0.13 0.00 0.56 0.00 0.03 0.04 0.06 0.00 0.38
0.05 0.03 0.20 0.14 0.39 0.00 0.19 0.00 0.03 0.00 0.00 0.00 0.04
0.04 0.08 0.00 0.11 0.04 0.08 0.00 0.00 0.12 0.00 0.24 0.17 0.09
0.00 0.03 0.00 0.03 0.00 0.07 0.00 0.06 0.00 0.15 0.04 0.00 0.00
0.07 0.00 0.07 0.00 0.00 0.07 0.14 0.09 0.00 0.00 0.11 0.42 0.04
0.00 0.00 0.03 0.03 0.00 0.33 0.03 0.25 0.00 0.03 0.00 0.04 0.00
0.23 0.03 0.03 0.06 0.07 0.00 0.22 0.00 0.04 0.00 0.16 0.00 0.20
0.00 0.00 0.03 0.03 0.00 0.12 0.07 0.11 0.00 0.03 0.06 0.15 0.00
0.11 0.00 0.00 0.00 0.00 0.23 0.12 0.03 0.00 0.03 0.03 0.03 0.00
0.12 0.00 0.04 0.07 0.00 0.13 0.00 0.15 0.00 0.10 0.04 0.00 0.00
0.00 0.00 0.05 0.03 0.00 0.00 0.04 0.03 0.00 0.36 0.00 0.05 0.03
0.03 0.07 0.00 0.00 0.00 0.06 0.07 0.00 0.08 0.00 0.06 0.05 0.08
0.00 0.46 0.03 0.07 0.00 0.00 0.00 0.03 0.63 0.05 0.04 0.00 0.03
0.17 0.00 0.03 0.03 0.00 0.04 0.03 0.06 0.04 0.09 0.00 0.09 0.00
0.00 0.00 0.00 0.06 0.00 0.37 0.12 0.06 0.03 0.04 0.03 0.00 0.04
0.00 0.00 0.00 0.00 0.00 0.03 0.00 0.02 0.03 0.03 0.13 0.06 0.11
0.08 0.05 0.03 0.03 0.00 0.09 0.15 0.03 0.06 0.04 0.04 0.06 0.08
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
0
1
2
4
6
7
8
9
10
11
12
13
14
ZFS
ZF30

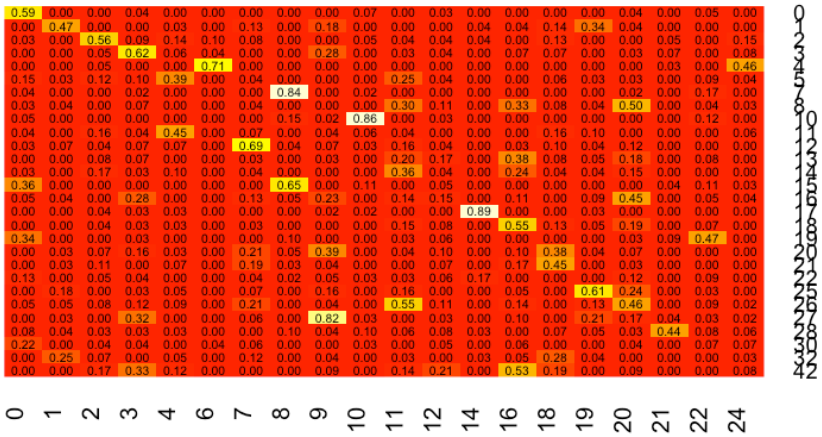

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

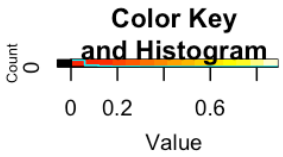

ZF60
ZF50

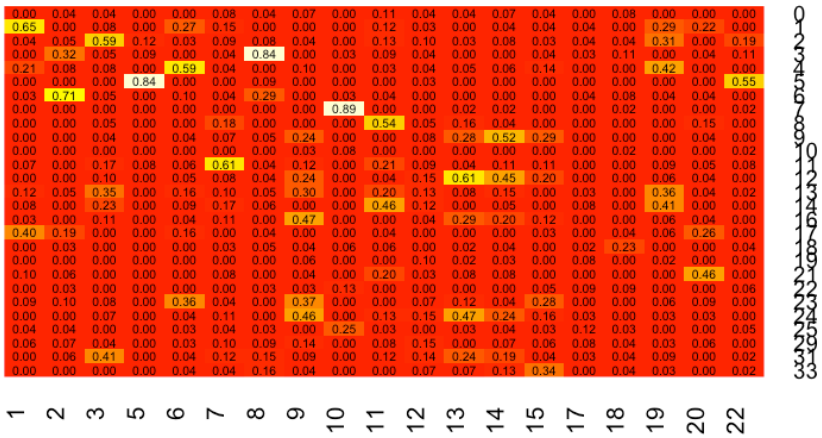

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value

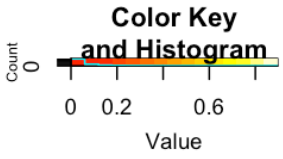

ZF75
ZFS

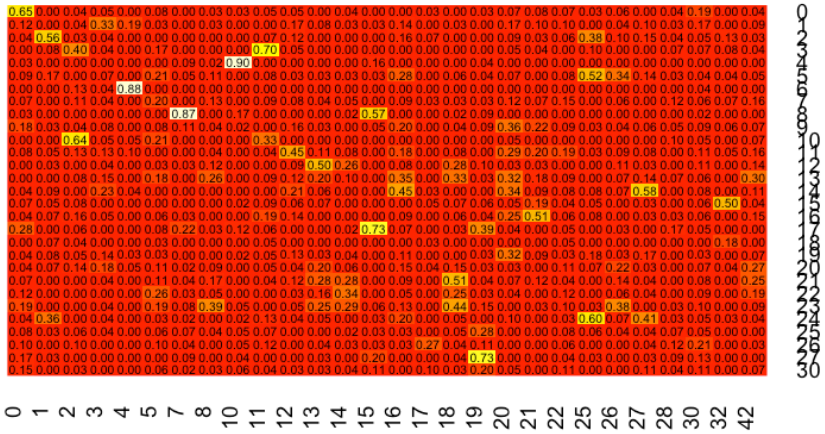

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

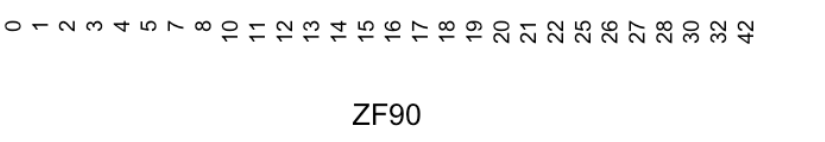

ZF90
ZF60

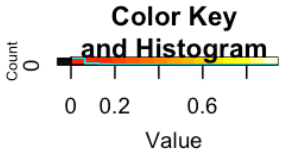

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

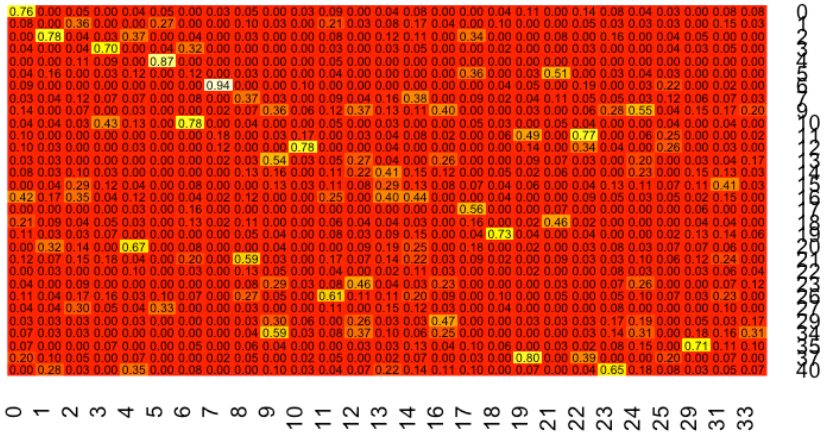

ZFB
ZF75

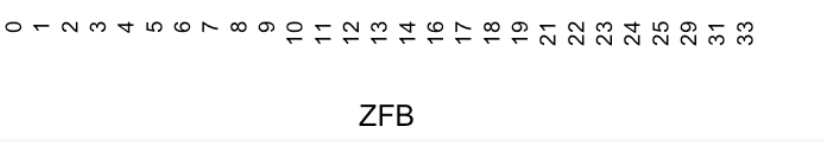













```
# stages1=c('zf6s','zf14s','zf20hpf') stages2=c('zf10s','zf18s')
# mod_rm2=low.connection(G_int2[stages1], stages=stages1, thres=0.2, ret_kp=F)
# mod_rm3=low.connection(G_int2[stages2], stages=stages2, thres=0.2, ret_kp=F)

# modules_rm=intersect(mod_rm1,union(mod_rm2,mod_rm3)) G_int_use <- list() for(i
# in 1:(length(stages)-1)){ stage=stages[i] stage_next=stages[i+1]
# G_cor_stage=G_int[[stage]]
# G_int_use[[stage]]=G_cor_stage[mod_kp[[stage_next]],mod_kp[[stage]]] }
```

Connect modules using the overlap scores calculated above

Build tables that record potential connections

For each module, find its most overlaped module in each of the two previous stages. Only modules with >22.5% overlaps are taken into account.

```
## for each module at one stage, want to find max correlated one in the two
## previous stages
connect_module <- function(thres1 = 0.15, thres2 = 0.25, G_cor_use, G_cor_use2) {
    G_connect <- list()
    for (i in 1:(length(stages) - 1)) {
        stage = stages[i]
        stage_next = stages[i + 1]
        G_cor_stage = G_cor_use[[stage]]
        Max_pre = apply(G_cor_stage, 1, order)
        Max_pre_ind = Max_pre[dim(Max_pre)[1], ]
        Max_pre_M = colnames(G_cor_stage)[Max_pre_ind]
        Max_value = apply(G_cor_stage, 1, max)
        has_pre_ind = which(Max_value > thres1)
        has_pre_M = rownames(G_cor_stage)[has_pre_ind]
        if (i == 1) {
            G_connect[[stage_next]] = data.frame(matrix(NA, nrow = 1, ncol = dim(G_cor_stage)[1]),
                row.names = stage)
            colnames(G_connect[[stage_next]]) = rownames(G_cor_stage)
            G_connect[[stage_next]][, has_pre_M] = Max_pre_M[has_pre_ind]
            G_connect[[stage_next]] = G_connect[[stage_next]][, has_pre_M]
        } else {
            stage_pre = stages[i - 1]
            G_cor_stage2 = G_cor_use2[[stage_pre]]
            all_M = union(rownames(G_cor_stage2), rownames(G_cor_stage))
            G_connect[[stage_next]] = data.frame(matrix(NA, nrow = 2, ncol = length(all_M)),
                row.names = c(stage, stage_pre))
            colnames(G_connect[[stage_next]]) = all_M
            G_connect[[stage_next]][1, has_pre_M] = Max_pre_M[has_pre_ind]
            G_cor_stage = G_cor_use2[[stage_pre]]
            Max_pre = apply(G_cor_stage, 1, order)
            Max_pre_ind = Max_pre[dim(Max_pre)[1], ]
            Max_pre_M = colnames(G_cor_stage)[Max_pre_ind]
            Max_value = apply(G_cor_stage, 1, max)
            has_pre_ind = which(Max_value > thres2)
            has_pre_M2 = rownames(G_cor_stage)[has_pre_ind]
            G_connect[[stage_next]][2, has_pre_M2] = Max_pre_M[has_pre_ind]
            G_connect[[stage_next]] = G_connect[[stage_next]][, union(has_pre_M,
                has_pre_M2)]
        }
    }
    return(G_connect)
}
G_int_connect = connect_module(G_cor_use = G_int, G_cor_use2 = G_int2, thres1 = 0.225,
    thres2 = 0.225)
```

Build an adjacency matrix to record the final connections between modules

We start from modules in the oldest stage (6-somites). Each module is first connected to its most overlaped module in the immediate previous stage. If no potential connection is recorded (in `G_int_connect`) for the immediate previous stage, it will then be connected to the module recorded for the stage earlier (if there is one). When the overlap between a module and its most overlapped module in the immediate previous stage is less than 35%, and at the same time it has more than 50% overlap with its most overlapped module two stages earlier, we then directly connect this module to the more previous module, and cut its connection to the one in the immidiate previous stage (this case didn't occur here).

```
build_netM <- function(G_connect, G_cor_use, G_cor_use2, thres = NULL, thres_pre = NULL) {
    nodes_names = c()
    for (i in 1:(length(stages) - 1)) {
        stage = stages[i + 1]
        G_ans = G_connect[[stage]]
        nodes_names = union(nodes_names, paste0(stage, "_", colnames(G_ans)))
        nodes_names = union(nodes_names, paste0(stages[i], "_", G_ans[stages[i],
```

```
            which(!is.na(G_ans[stages[i], ]))]))
        if (i > 1) {
            nodes_names = union(nodes_names, paste0(stages[i - 1], "_", G_ans[stages[i -
                1], which(!is.na(G_ans[stages[i - 1], ]))]))
        }
    }
    num_nodes = length(nodes_names)
    net_M = matrix(0, ncol = num_nodes, nrow = num_nodes)
    rownames(net_M) = nodes_names
    colnames(net_M) = nodes_names

    for (i in 1:(length(stages) - 1)) {
        stage_pre = stages[i]
        stage = stages[i + 1]
        G_ans = G_connect[[stage]]
        for (j in colnames(G_ans)) {
            to_name = paste0(stage, "_", j)
            if (!is.na(G_ans[stage_pre, j])) {
                from_M = G_ans[stage_pre, j]
                from_name = paste0(stage_pre, "_", from_M)
                ## get the correlation score to put in the connection matirx
                net_M[from_name, to_name] = G_cor_use[[stage_pre]][j, from_M]
            }
            if (i != 1) {
                stage_pre2 = stages[i - 1]
                if (!is.na(G_ans[stage_pre2, j])) {
                  from_M2 = G_ans[stage_pre2, j]
                  from_name2 = paste0(stage_pre2, "_", from_M2)
                  if (is.na(G_ans[stage_pre, j])) {
                    net_M[from_name2, to_name] = G_cor_use2[[stage_pre2]][j, from_M2]
                  } else if (!is.null(thres)) {
                    G_cor = G_cor_use[[stage_pre]][j, from_M]
                    G_cor_pre = G_cor_use2[[stage_pre2]][j, from_M2]
                    if (G_cor < thres && G_cor_pre > thres_pre) {
                      print(paste0("add ", from_name2, " to ", to_name))
                      net_M[from_name2, to_name] = G_cor_use2[[stage_pre2]][j, from_M2]
                      print(paste0("delete ", from_name, " to ", to_name))
                      net_M[from_name, to_name] = 0
                    }
                  }
                }
            }
        }
    }
    return(net_M)
}

net_int = build_netM(G_int_connect, G_int, G_int2, thres = 0.35, thres_pre = 0.5)
```

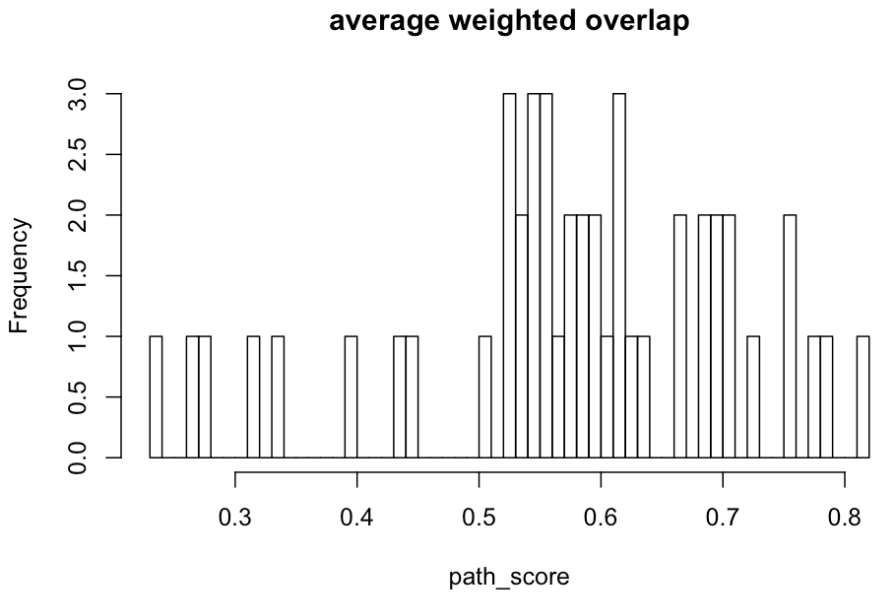
Trim path with poor quality

```
get_downstream <- function(net_M, start_M, exclude = c("")) {
    all_ds = c(start_M)
    M_ds = colnames(net_M)[which(net_M[start_M, ] > 0)]
    M_ds = M_ds[which(!M_ds %in% exclude)]
    if (length(M_ds) > 0) {
        all_ds = unique(c(all_ds, M_ds))
        for (M_d in M_ds) {
            all_ds = unique(c(all_ds, get_downstream(net_M, M_d, exclude = exclude)))
        }
    }
    return(all_ds)
}

get_upstream <- function(net_M, start_M, exclude = c(""), mean_score = F, start_score = 0,
    start_num_ans = 0) {
    all_as = c(start_M)
    M_as = rownames(net_M)[which(net_M[, start_M] > 0)]
    M_as = M_as[which(!M_as %in% exclude)]
```

```
    num_ans = start_num_ans
    tot_score = start_score
    if (length(M_as) > 0) {
        all_as = unique(c(all_as, M_as))
        num_ans = num_ans + length(M_as)
        # print(num_ans)
        tot_score = tot_score + sum(net_M[M_as, start_M])
        # print(tot_score)

        for (M_a in M_as) {
            if (mean_score) {
                in_result_list = get_upstream(net_M, M_a, exclude = exclude, mean_score = T,
                  start_score = tot_score, start_num_ans = num_ans)
                all_as = unique(c(all_as, in_result_list$upstream))
                # print(all_as) print(in_result_list$score)
                tot_score = in_result_list$score[1]
                num_ans = in_result_list$score[2]
            } else {
                all_as = unique(c(all_as, get_upstream(net_M, M_a, exclude = exclude)))
            }
        }
    }
    if (mean_score) {
        return_list = list()
        return_list$upstream = all_as
        return_list$score = c(tot_score, num_ans)
        return(return_list)
    } else {
        return(all_as)
    }
}

calc_path_qual <- function(net_M, path = "all", exclude = c("")) {
    ## calculate the mean overlap level along the path end at the specified node(s)
    if (path == "all") {
        end_nodes = rownames(net_M)[which(apply(net_M, 1, max) == 0)]
    } else {
        end_nodes = path
    }
    score_vec = c(1:length(end_nodes)) * 0
    names(score_vec) = end_nodes
    for (node in end_nodes) {
        node_score = get_upstream(net_M, node, mean_score = T, exclude = exclude)
        score_vec[node] = node_score$score[1]/node_score$score[2]
    }
    return(score_vec)
}
```

Calculate the average overlap score along each chain of connected gene modules

```
path_score = calc_path_qual(net_int)
hist(path_score, breaks = 50, main = "average weighted overlap")
```



Keep only the paths with >0.44 average weighted overlap.

Most of the path with <0.44 average overlap were short or consist of either ubiquitous or lowly expressed genes.

```
# ## inspect the low quality module chains
# end_nodes_bad=names(path_score[path_score<0.44]) for(node in end_nodes_bad){
# #print(get_upstream(net_int,node)) bad_path=get_upstream(net_int,node)
# bad_tbl=matrix(nrow=20,ncol=length(bad_path)) colnames(bad_tbl)=bad_path
# for(bad_node in bad_path){ stage=unlist(strsplit(bad_node,'_'))[1]
# m=unlist(strsplit(bad_node,'_'))[2]
# bad_tbl[,bad_node]=as.character(zf_top[[stage]][1:20,paste0('Module.',m)]) }
# print(bad_tbl) }
end_nodes_good = names(path_score[path_score >= 0.44])
all_nodes_good = c()
for (node in end_nodes_good) {
    all_nodes_good = c(all_nodes_good, get_upstream(net_int, node))
}
all_nodes_good = unique(all_nodes_good)

net_int_good = net_int[all_nodes_good, all_nodes_good]
```

Save connected module information for overlaying on URD tree

For each module at the end (oldest developmental stage) of a connected chain, find all its upstream modules, and store them as an entry in one list

```
all_end_nodes = rownames(net_int_good)[which(apply(net_int_good, 1, sum) == 0)]
all_lineages <- list()
for (end_node in all_end_nodes) {
    up_nodes = get_upstream(net_int_good, end_node)
    path_tbl = data.frame(matrix(nrow = 25, ncol = 2 * length(up_nodes)), stringsAsFactors = F)
    name_col = rep(up_nodes, each = 2)
    name_col[c(1:length(up_nodes)) * 2] = paste0(name_col[c(1:length(up_nodes)) *
        2], "_Weight")
    colnames(path_tbl) = name_col
    for (node in up_nodes) {
        stage = unlist(strsplit(node, "_"))[1]
        m = unlist(strsplit(node, "_"))[2]
        path_tbl[, node] = as.character(zf_top[[stage]][1:25, paste0("Module.", m)])
```

```
        path_tbl[, paste0(node, "_Weight")] = (zf_top[[stage]][1:25, paste0("Weights.",
            m)])
    }
    all_lineages[[end_node]] = path_tbl
}
# save(all_lineages,file='./Module_tree/module_lineages.Robj')
```

Print out modules in a few lineages

```
for (lineage in names(all_lineages)[1:3]) {
    print(all_lineages[[lineage]])
}
```

```
##            ZFS_11 ZFS_11_Weight         ZF50_13 ZF50_13_Weight
## 1           FOSAB     4.8460328           FOSAB     10.0433512
## 2           JUNBA     2.5118439            ATF3      5.6183320
## 3  CABZ01070258.1     1.7785344           JUNBA      4.5060923
## 4      CU019646.2     1.7278177           KLF2B      2.5415479
## 5         CDKN1CB     1.5890029  CABZ01070258.1      2.5151719
## 6            ATF3     1.3202577         CDKN1CB      2.1350717
## 7            CDK1     1.2771422            BTG2      1.9640439
## 8         PFKFB4B     1.1978317        GADD45GA      1.9238977
## 9            RRM2     1.1574811      CU019646.2      1.6461276
## 10          FBXO5     1.1549479 SI:DKEY-27I16.2      1.5461727
## 11          FBXO2     1.1264328          HIGD1A      1.3749177
## 12           TPX2     1.1175181             JUN      1.2887396
## 13            JUN     1.0993639           SEPH1      1.2025177
## 14        DYNLL2A     1.0274270           H2AFX      1.1061813
## 15          ACTB1     0.9933374  SI:DKEY-68O6.5      1.0981300
## 16         CDC14B     0.9278185        GADD45BA      1.0713179
## 17           BTG2     0.8935460          ABRACL      1.0149452
## 18        RASL11B     0.8654750       ZGC:92818      1.0033499
## 19          PTMAA     0.8427796           ACTB1      0.9978422
## 20      HIST2H2AB     0.8257449         ANGPTL4      0.9829408
## 21        FAM46BA     0.8125369          TUBB4B      0.9746121
## 22           MIBP     0.7843264            ID2A      0.9597552
## 23           RRP1     0.7682849            SDC4      0.9298233
## 24         ZC3H13     0.7650125           PPRC1      0.9292362
## 25          STOX1     0.7551083 SI:DKEY-15J16.6      0.9011777
##          ZFDOME_10 ZFDOME_10_Weight
## 1            FOSAB        2.6323789
## 2              JUN        1.4138799
## 3  SI:DKEY-261J4.5        1.3396949
## 4            KLF2B        1.2584963
## 5           SNAI1A        1.2175813
## 6             NOTO        1.2081713
## 7            ACTB1        1.1846537
## 8              VOX        1.1749096
## 9          CDKN1CA        1.1710203
## 10          FBXO5        1.1680097
## 11         SHISA2        1.1214234
## 12           TPX2        1.1153018
## 13          KLF17        1.0880430
## 14        CASP8AP2        1.0716860
## 15           H2AFX        1.0570557
## 16           ACP5A        1.0406903
## 17           TOB1A        1.0132369
## 18           APOEB        0.9612119
## 19          DYNLL1        0.9523646
## 20            LIG1        0.9433761
## 21           TBX16        0.9375308
## 22           JUNBA        0.9334967
## 23            P4HB        0.9211129
## 24           APELA        0.9201772
## 25         FAM46BA        0.9033595
##       ZF75_22 ZF75_22_Weight    ZF60_24 ZF60_24_Weight         ZFS_20
## 1        PFN1      6.4845322        LYE      6.3134688            LYE
## 2         LYE      6.2444244    SULT6B1      5.1325590        SULT6B1
## 3        KRT4      6.0523372       PFN1      2.7380534          KRT18
## 4       KRT18      4.9689508     F11R.1      2.6250161        TAGLN2
```

```
## 5     SULT6B1      3.9283869      KRT18      2.3985611           KRT4
## 6        KRT8      3.6257115       KRT4      2.1996187         F11R.1
## 7        KRT5      3.1671185     TAGLN2      2.0179354           PFN1
## 8      F11R.1      2.9446991       KRT8      1.9652621          CLDNE
## 9  ZGC:101000      2.1973960       CNN2      1.5931257           KRT8
## 10  HIST2H2AB      1.4229633      EPCAM      1.5851842         VGLL4L
## 11     TAGLN2      1.2687562 ZGC:101000      1.4078778           TIFA
## 12    ARHGAP5      1.2459263       SSR4      1.3018267 CABZ01070258.1
## 13 ZGC:193505      1.0699770       KRT5      1.2973653     ZGC:101000
## 14    SNRNP40      1.0497005      PDIA6      1.1518407         ACKR3B
## 15       CD9B      0.9676557      GLULB      1.0770835           HER6
## 16     TMSB4X      0.9511427  HIST2H2AB      1.0432061      HSD17B12A
## 17       CNN2      0.9422662    ARHGDIA      1.0306307           FKBP7
## 18       JUPA      0.8990241     SOX11A      1.0291283           CNN2
## 19      SEPW1      0.8949789       JUPA      0.9976995      HIST2H2AB
## 20 ZGC:109934      0.8872325       CD9B      0.9608146          PTGDSB
## 21      EPCAM      0.8568967    CRABP2B      0.9488314           BTG1
## 22     ABRACL      0.8540566    PFKFB4B      0.9389112       ZGC:92066
## 23  ZGC:92242      0.8341328      SEPW1      0.9125960           GID8B
## 24      CLDNE      0.8321923     SEC61B      0.8449103     ZGC:109934
## 25       TPM3      0.8319476   TP53INP2      0.8364075           CD9B
##    ZFS_20_Weight    ZF50_14 ZF50_14_Weight           ZF30_10
## 1      1.7298615        LYE       6.305011               LYE
## 2      1.7090349    SULT6B1       4.212302            TAGLN2
## 3      1.7075829      KRT18       4.086774             KRT18
## 4      1.6667962       KRT8       2.965524            SULT6B1
## 5      1.3847256     TAGLN2       2.837602              PFN1
## 6      1.3365582       KRT4       2.702960              KRT8
## 7      1.2568417       PFN1       2.276298           ALOX5B.3
## 8      1.0860781     F11R.1       2.149443            CRABP2B
## 9      1.0858555       KRT5       2.012182             F11R.1
## 10     1.0087559      CLDNE       1.758102             VGLL4L
## 11     0.9562132     SOX11A       1.710053           MID1IP1A
## 12     0.9147088     VGLL4L       1.654682               KRT4
## 13     0.9047130       CNN2       1.631431              CLDNE
## 14     0.8331175      EPCAM       1.616913               XBP1
## 15     0.8164230 ZGC:101000       1.499036               KRT5
## 16     0.7944954    CRABP2B       1.486763              EPCAM
## 17     0.7823978        BLF       1.369434             SOX11A
## 18     0.7674159      TDGF1       1.198810               CNN2
## 19     0.7633843     LRRC59       1.184705                BLF
## 20     0.7580118 HSP90AA1.2       1.168386          ZGC:92818
## 21     0.7322261       XBP1       1.165778            STARD14
## 22     0.7201393   MID1IP1A       1.150966 SI:CH211-125O16.4
## 23     0.7022535       JUPA       1.135152         ZGC:101000
## 24     0.7019037       TIFA       1.100384               TIFA
## 25     0.6803540  ZGC:92818       1.058742              TDGF1
##    ZF30_10_Weight        ZFDOME_9 ZFDOME_9_Weight        ZFOBLONG_1
## 1       3.4650354          KRT18       2.8132192          MID1IP1A
## 2       2.6149203        MID1IP1A       2.0842485           STARD14
## 3       2.3328672             LYE       2.0211834           GADD45BB
## 4       2.0663134         SULT6B1       1.8280506              CLDNE
## 5       1.6636238         STARD14       1.5743479            ARL13A
## 6       1.5568584          TAGLN2       1.2423134               LYE
## 7       1.5448787            KRT8       1.1829682             CEBPB
## 8       1.5413767           EPCAM       1.1452402             KRT18
## 9       1.4830910          MPLKIP       1.0769785             KLF17
## 10      1.4682137          VGLL4L       1.0119617            MPLKIP
## 11      1.4134182            XBP1       1.0113424           SULT6B1
## 12      1.4046074          F11R.1       1.0098688             GRHL3
## 13      1.3470858 SI:DKEY-261J4.5       0.9390450            SOX19A
## 14      1.3134039            BTG1       0.9310279           GADD45BA
## 15      1.2429403           ACTB1       0.9154082              FTR82
## 16      1.2130029           FTR82       0.8343180            ACKR3B
## 17      1.1861783        ALOX5B.3       0.8190791             ACTB1
## 18      0.9255266           TDGF1       0.8006291 SI:CH211-125O16.4
## 19      0.9050472           CLDNE       0.7920283             CLDN7B
## 20      0.8951153            PFN1       0.7837560          MARCKSL1B
## 21      0.8592334           PNRC2       0.7760069              KLF2A
## 22      0.8029567          HIRIP3       0.7459742               TIFA
## 23      0.8005009            KRT4       0.7219277               KRT8
## 24      0.7903775            KRT5       0.7212864            APLNRA
```

```
## 25      0.7756159            CNN2       0.7086680              HER6
##    ZFOBLONG_1_Weight  ZFHIGH_2 ZFHIGH_2_Weight
## 1          1.4455331  MID1IP1A       1.7556321
## 2          1.3764646   STARD14       1.0398555
## 3          0.9961665     KRT18       0.9338936
## 4          0.9954582     KLF17       0.9288900
## 5          0.9657695  GADD45BB       0.9272871
## 6          0.9181615     CCND1       0.8893895
## 7          0.9175384   CDKN1CA       0.7937746
## 8          0.9138730      BTG1       0.7825260
## 9          0.8412915  GADD45BA       0.7766364
## 10         0.7819312    MPLKIP       0.7600799
## 11         0.7783169     LRWD1       0.7580100
## 12         0.6794846      NET1       0.7419168
## 13         0.6595933     CLDNE       0.7009932
## 14         0.6463019    ARL13A       0.6945497
## 15         0.6398491     GRHL3       0.6527599
## 16         0.6359219 MARCKSL1B       0.6400235
## 17         0.6319915    ACKR3B       0.6168704
## 18         0.6090443    SOX19A       0.6160761
## 19         0.5997524      IRX7       0.5971268
## 20         0.5912220   RASSF7B       0.5937983
## 21         0.5807648    MALAT1       0.5834228
## 22         0.5776952      SOX3       0.5761636
## 23         0.5604718     FOXI1       0.5652665
## 24         0.5563380     CEBPB       0.5439890
## 25         0.5249795    CLDN7B       0.5437657
##       ZF90_5 ZF90_5_Weight    ZF75_2 ZF75_2_Weight   ZF60_4 ZF60_4_Weight
## 1       XBP1      4.355823      XBP1      4.242558     XBP1      4.051199
## 2       MNS1      2.885853   CDKN1CA      3.454459     NOTO      3.602728
## 3   NKX1.2LA      2.870412      NOTO      3.292378  CDKN1CA      3.173418
## 4     TUBA1A      2.635527      INSB      2.686459      CHD      2.913641
## 5     SOX11A      2.493386      ADMP      2.604827     ADMP      2.843297
## 6       NOTO      2.420971      CST3      2.556160     CST3      2.237629
## 7    CITED4B      2.399929      NTD5      2.554613     INSB      1.932917
## 8       ADMP      1.933957    SEC61G      2.533509    FOXA2      1.814984
## 9      FOXA2      1.793524      SSR2      2.443110   MAGI1B      1.766576
## 10      SHHB      1.771029      SSR3      2.344111    P4HA2      1.706590
## 11 ZGC:92066      1.753219        TA      2.340881  CITED4B      1.633882
## 12      LFT2      1.669816     P4HA2      2.295235       TA      1.571454
## 13     FOXD3      1.620735       CHD      2.170651     MIDN      1.503120
## 14    TMSB4X      1.577562      SSR4      2.149654     SHHA      1.498016
## 15      HER6      1.531934 SERPINH1B      2.082437  RASL11B      1.480979
## 16     PTMAA      1.447509      FOXA      2.072362     FOXA      1.457629
## 17     CDH11      1.417092     PLOD1A      1.949791     NET1      1.431948
## 18     FGF8A      1.377407    SEC61B      1.837591  BCL2L12      1.409584
## 19     P4HA2      1.366734      NOG1      1.827007   ARL4AB      1.398962
## 20    POU5F3      1.284828    TWIST2      1.821286   SEC61G      1.382941
## 21     TCF3B      1.258434     HSPA5      1.785671 GADD45BA      1.371876
## 22      SSR3      1.231777     FOXA2      1.779097    FOXA3      1.359905
## 23   CDKN1CA      1.217085    ZFAND5A      1.769714     NTD5      1.302313
## 24    SEC61G      1.143285     LRRC59      1.745463   FSCN1A      1.242413
## 25      SSR2      1.109916      QKIA      1.701277    DACT2      1.212931
##       ZFS_4 ZFS_4_Weight   ZF50_12 ZF50_12_Weight          ZF30_4
## 1      NOTO    2.6821919      NOTO       5.015222            NOTO
## 2      XBP1    1.9434436      XBP1       4.132969            XBP1
## 3        TA    1.9375870       CHD       3.130884           FOXD3
## 4   CDKN1CA    1.7392100    FSCN1A       3.098186          FSCN1A
## 5      HES6    1.4330684      ADMP       3.088155          HIGD1A
## 6      WNT11   1.3524305 RASGEF1BA       2.791966       RASGEF1BA
## 7  RASGEF1BA    1.3160654        TA       2.589643             CHD
## 8       ADMP    1.2969467     FOXD3       1.915100           TDGF1
## 9   BCL2L12    1.2035805    ARL4AB       1.889387            ADMP
## 10     FOXD3    1.1784662     APELA       1.854695            TIFA
## 11    FSCN1A    1.1267671      NET1       1.847418              TA
## 12   RASL11B    1.1074693     WNT11       1.759278           HSPB1
## 13       CHD    1.0921043    HIGD1A       1.730274         GADD45BA
## 14      SSR4    0.9589195    MAGI1B       1.722602           CX43.4
## 15    HIGD1A    0.9571933     DDIT4       1.703887           MAGI1B
## 16     DDIT4    0.9337054  GADD45BA       1.701777        HIST2H2AB
## 17     P4HA2    0.9221352      CST3       1.653110             HES6
## 18     FGF8A    0.9063344   CDKN1CA       1.639709          CITED4B
```

```
## 19      ETV4    0.8992707      HES6       1.604342             CST3
## 20     TBX16    0.8990574     P4HA2       1.486225            RPS20
## 21     UBE2C    0.8803425   CITED4B       1.454920         MYL12.1
## 22   DYNLL2A    0.8233651    ZSWIM5       1.433536            LRWD1
## 23   CITED4B    0.7989774     TDGF1       1.392326 SI:CH211-152C2.3
## 24    CX43.4    0.7982470      BTG2       1.360730             RPL6
## 25      NET1    0.7788877     TBX16       1.310208            APELA
##    ZF30_4_Weight ZFDOME_12 ZFDOME_12_Weight          ZFHIGH_6
## 1      2.8999539      NOTO        2.3228672              NOTO
## 2      2.1892567      XBP1        2.1086143             WNT11
## 3      2.1398977 RASGEF1BA        2.0225759               GSC
## 4      2.1046486       CHD        1.9713327             FOXD3
## 5      1.7575036     FOXD3        1.8251680              SDF4
## 6      1.7293570    FSCN1A        1.5128735              EZRB
## 7      1.7182033     DACT2        1.3952872           STARD14
## 8      1.6896381    HIGD1A        1.3701726              FOXA
## 9      1.4981775       GSC        1.3188713             TBX16
## 10     1.4290808      ADMP        1.1757926            SOX11A
## 11     1.2663629      FOXA        1.1187955          RASGEF1BA
## 12     1.1656576     LRWD1        1.1053462           PARD6GB
## 13     1.1543488     DDIT4        0.9690360             CEBPB
## 14     1.1221845     SRP19        0.9487148              NOG1
## 15     1.1092605 HIST2H2AB        0.9484454              PLK3
## 16     1.0837475     RPL38        0.9348785 SI:DKEY-108K21.14
## 17     1.0811422    ZNFL2A        0.9247886              BTG1
## 18     1.0364812      DENR        0.9235584          ZGC:92066
## 19     1.0195759      CST3        0.8959908            TOLLIP
## 20     1.0090414    TMSB4X        0.8901142           TSPAN15
## 21     1.0006236  GADD45BA        0.8874434          ZDHHC16A
## 22     0.9801442    BIRC5A        0.8837842             FZD8B
## 23     0.9786745     TDGF1        0.8703037   SI:DKEY-228B2.6
## 24     0.9772427     KPNA2        0.8644927              OTX1B
## 25     0.9751178     TTC31        0.8553763              MYH9A
##    ZFHIGH_6_Weight
## 1        0.8782326
## 2        0.8033856
## 3        0.7751899
## 4        0.5713401
## 5        0.5033481
## 6        0.4884126
## 7        0.4877907
## 8        0.4844816
## 9        0.4839370
## 10       0.4811804
## 11       0.4520987
## 12       0.4438361
## 13       0.4427354
## 14       0.4345067
## 15       0.4343383
## 16       0.4289524
## 17       0.4270979
## 18       0.4161802
## 19       0.4051465
## 20       0.4050952
## 21       0.4046677
## 22       0.3984694
## 23       0.3960544
## 24       0.3909067
## 25       0.3891016
```

For modules that are in the same connected chain, sum up their levels in each cell to represent the expression of that lineage program. This results is a lineage by cell matrix

```
lineage_exp <- function(DS_C_use, DS_G_use, net_int_good, all_lineages, all_end_nodes = NULL) {
    if (is.null(all_end_nodes)) {
        all_end_nodes = rownames(net_int_good)[which(apply(net_int_good, 1, sum) ==
            0)]
    }
```

```
    stages = names(DS_C_use)
    all_cells = c()
    all_genes = c()
    for (stage in stages) {
        C_use = DS_C_use[[stage]]
        all_cells = c(all_cells, colnames(C_use))
        G_use = DS_G_use[[stage]]
        all_genes = c(all_genes, rownames(G_use))
    }

    all_genes = unique(all_genes)
    all_Ms = rownames(net_int_good)
    allM_allCell = data.frame(matrix(0, ncol = length(all_cells), nrow = length(all_Ms)),
        row.names = all_Ms)
    allGene_allM = data.frame(matrix(0, ncol = length(all_Ms), nrow = length(all_genes)),
        row.names = all_genes)
    colnames(allM_allCell) = all_cells
    colnames(allGene_allM) = all_Ms
    ## look stage by stage, fill in the expression matrix with MAX NORMALIZED gene
    ## module expression
    for (stage in stages) {
        G_use = DS_G_use[[stage]]
        G.max = apply(G_use, 2, max)
        G_norm = sweep(G_use, 2, G.max, "/")  ## now each module's top gene has weight 1
        colnames(G_norm) = paste0(stage, "_", colnames(G_norm))
        M_use = intersect(colnames(G_norm), all_Ms)

        C_use = DS_C_use[[stage]]
        C.max = apply(C_use, 1, max)
        C_norm = sweep(C_use, 1, C.max, "/")
        rownames(C_norm) = paste0(stage, "_", rownames(C_norm))

        if (length(M_use) > 0) {
            ## fill in gene matrix
            allGene_allM[rownames(G_norm), M_use] = G_norm[rownames(G_norm), M_use]
            ## fill in cell matrix
            allM_allCell[M_use, colnames(C_use)] = C_norm[M_use, colnames(C_use)]
        }
    }

    lineage_cell = data.frame(matrix(0, ncol = length(all_cells), nrow = length(all_end_nodes)),
        row.names = all_end_nodes, stringsAsFactors = F)
    colnames(lineage_cell) = all_cells

    # matrix to use: allM_allCell
    for (lin in all_end_nodes) {
        lin_M = colnames(all_lineages[[lin]])[c(T, F)]
        if (length(setdiff(lin_M, all_Ms)) == 0) {
            ## sum up and add
            lineage_cell[lin, ] = apply(allM_allCell[lin_M, colnames(lineage_cell)],
                2, sum)
        } else {
            print(paste(lin, "has module(s) that are not in the table"))
        }
    }
    return(list(lineageXcell = lineage_cell, allMXallCell = allM_allCell, allGeneXallM = allGene_allM))
}

lineage_module_list = lineage_exp(zf_use$C, zf_use$G, net_int_good, all_lineages,
    all_end_nodes = all_end_nodes)
lineage_cell = lineage_module_list$lineageXcell
allM_allCell = lineage_module_list$allMXallCell
allGene_allM = lineage_module_list$allGeneXallM
```

Define functions for visualizing the tree

```
clan_coord <- function(net_int_good, node_start, y.names = stages) {
    y = rev(1:length(y.names))
    names(y) = y.names
```

```
    nodes_in_clan <- get_downstream(net_int_good, node_start)
    clan_net <- net_int_good[nodes_in_clan, nodes_in_clan]
    end_nodes <- rownames(clan_net)[which(apply(clan_net, 1, sum) == 0)]
    coords = matrix(nrow = length(nodes_in_clan), ncol = 2)
    rownames(coords) = nodes_in_clan
    colnames(coords) = c("x", "y")
    m.stages = unlist(lapply(nodes_in_clan, function(x) unlist(strsplit(x, "_"))[1]))
    ys = y[m.stages]
    coords[, "y"] = ys
    if (length(end_nodes) == 1) {
        xs = rep(0, length(nodes_in_clan))
        names(xs) = nodes_in_clan
        coords[, "x"] = xs[rownames(coords)]
        return(coords)
    } else {
        up_nodes <- list()
        for (node in end_nodes) {
            up_nodes[[node]] = get_upstream(clan_net, node)
        }
        end_i = end_nodes[1]
        end.nodes_added <- c(end_i)
        while (length(setdiff(end_nodes, end.nodes_added)) > 0) {
            num.comm = unlist(lapply(up_nodes[setdiff(end_nodes, end.nodes_added)],
                function(x) length(intersect(up_nodes[[node_start]], x))))
            end_i = names(which.max(num.comm))
            end.nodes_added <- c(end.nodes_added, end_i)
        }
        end.nodes_xs = 1:length(end_nodes)
        names(end.nodes_xs) = end.nodes_added

        num_branch <- apply(clan_net > 0, 1, sum)
        branch_nodes = names(num_branch)[which(num_branch > 1)]
        if (!node_start %in% branch_nodes) {
            branch_nodes = c(branch_nodes, node_start)
        }
        branch.nodes_xs = c()
        for (node in branch_nodes) {
            branch_ends = intersect(get_downstream(clan_net, node), end_nodes)
            branch.nodes_xs = c(branch.nodes_xs, mean(end.nodes_xs[branch_ends]))
        }
        names(branch.nodes_xs) = branch_nodes
        for (node in branch_nodes) {
            up_nodes[[node]] = get_upstream(clan_net, node)
        }
        anchor.nodes_xs = c(end.nodes_xs, branch.nodes_xs)
        nodes.added = c()
        xs.all = c()
        for (node in names(up_nodes)) {
            branch_up = intersect(up_nodes[[node]], branch_nodes)
            seg_nodes = setdiff(up_nodes[[node]], unique(unlist(up_nodes[setdiff(branch_up,
                node)])))
            xs.all = c(xs.all, rep(as.numeric(anchor.nodes_xs[node]), length(seg_nodes)))
            nodes.added = c(nodes.added, seg_nodes)
        }
        names(xs.all) = nodes.added
        xs.all[names(branch.nodes_xs)] = branch.nodes_xs
        coords[, "x"] = xs.all[rownames(coords)]
        return(coords)
    }
}
start_nodes <- colnames(net_int_good)[which(apply(net_int_good, 2, sum) == 0)]
all_sub_coords = list()
num_modules = c()
for (node in start_nodes) {
    all_sub_coords[[node]] = clan_coord(net_int_good, node)
    num_modules = c(num_modules, length(get_downstream(net_int_good, node)))
}
names(num_modules) = start_nodes
ordered_modules = names(sort(num_modules, decreasing = T))
combined_coords = all_sub_coords[[ordered_modules[1]]]
for (lin_ind in 2:length(ordered_modules)) {
```

```
    base_coord = max(combined_coords[, "x"]) + 1
    coord2bind = all_sub_coords[[ordered_modules[lin_ind]]]
    coord2bind[, "x"] = coord2bind[, "x"] + base_coord
    combined_coords = rbind(combined_coords, coord2bind)
}

library(igraph)
g <- graph.adjacency(net_int_good > 0)
edge_list = get.edgelist(g)
```

Save all information in a list

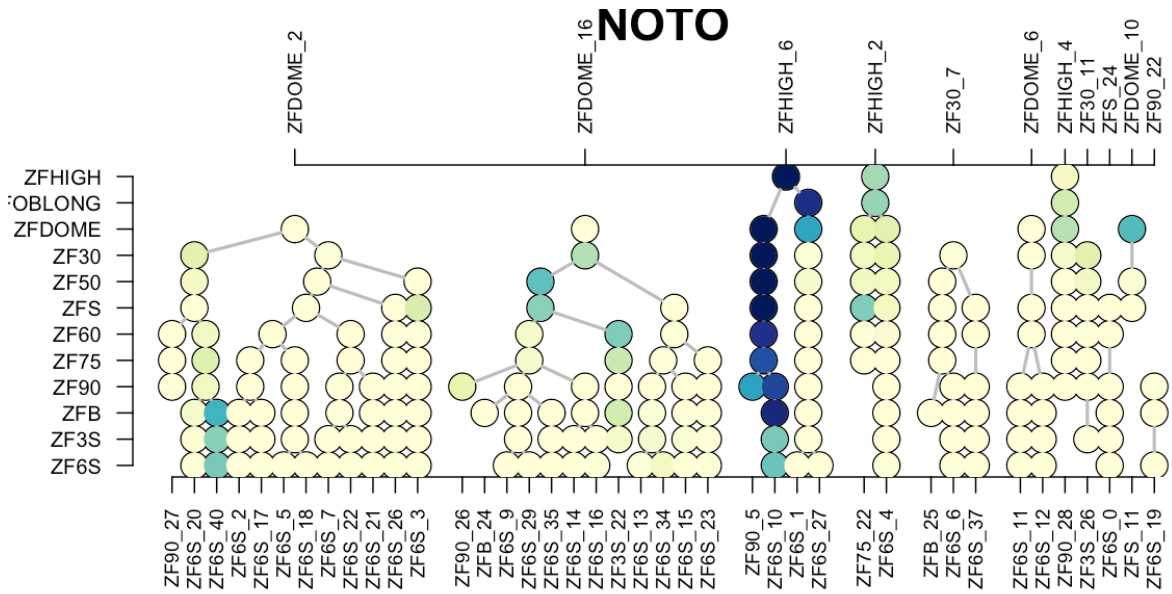
```
M.tree <- list(geneXmodule = allGene_allM, moduleXcell = allM_allCell, lineageXcell = lineage_cell,
    net_adj = net_int_good, coords = combined_coords, edge_list = edge_list, lineage_ident = all_lineages,
    top_genes = zf_top, ordered_stages = stages, roots = start_nodes, tips = all_end_nodes)
# saveRDS(M.tree, './ModuleTree201809.rds')
```

Plot gene weights on the module tree

```
library(RColorBrewer)
map2color <- function(x, pal, limits = NULL) {
    if (is.null(limits))
        limits = range(x)
    pal[findInterval(x, seq(limits[1], limits[2], length.out = (length(pal) + 1)),
        all.inside = TRUE)]
}
plot_MTree <- function(gene, M.Tree) {
    start_nodes = M.Tree$roots
    all_end_nodes = M.Tree$tips
    allGene_allM = M.Tree$geneXmodule
    combined_coords = M.Tree$coords
    edge_list = M.Tree$edge_list
    stages = M.Tree$ordered_stages
    cols = map2color(allGene_allM[gene, rownames(combined_coords)], pal = colorRampPalette(brewer.pal(9,
        "YlGnBu"))(200), limits = c(0, 1))
    par(mar = c(5, 4, 5, 2))
    plot(combined_coords[, "x"], combined_coords[, "y"], pch = 1, cex = 2.3, axes = F,
        xlab = "", ylab = "")
    title(main = gene, line = 4, cex.main = 1.8)
    axis(2, at = 1:length(stages), labels = rev(stages), las = 1, cex.axis = 0.8)
    axis(3, at = combined_coords[start_nodes, "x"], labels = start_nodes, las = 2,
        cex.axis = 0.75)
    axis(1, at = combined_coords[all_end_nodes, "x"], labels = all_end_nodes, las = 2,
        cex.axis = 0.75)
    for (i in 1:dim(edge_list)[1]) {
        lines(combined_coords[edge_list[i, ], "x"], combined_coords[edge_list[i,
            ], "y"], col = "gray", lwd = 2)
    }
    points(combined_coords[, "x"], combined_coords[, "y"], pch = 16, col = cols,
        cex = 2.2)
}
```

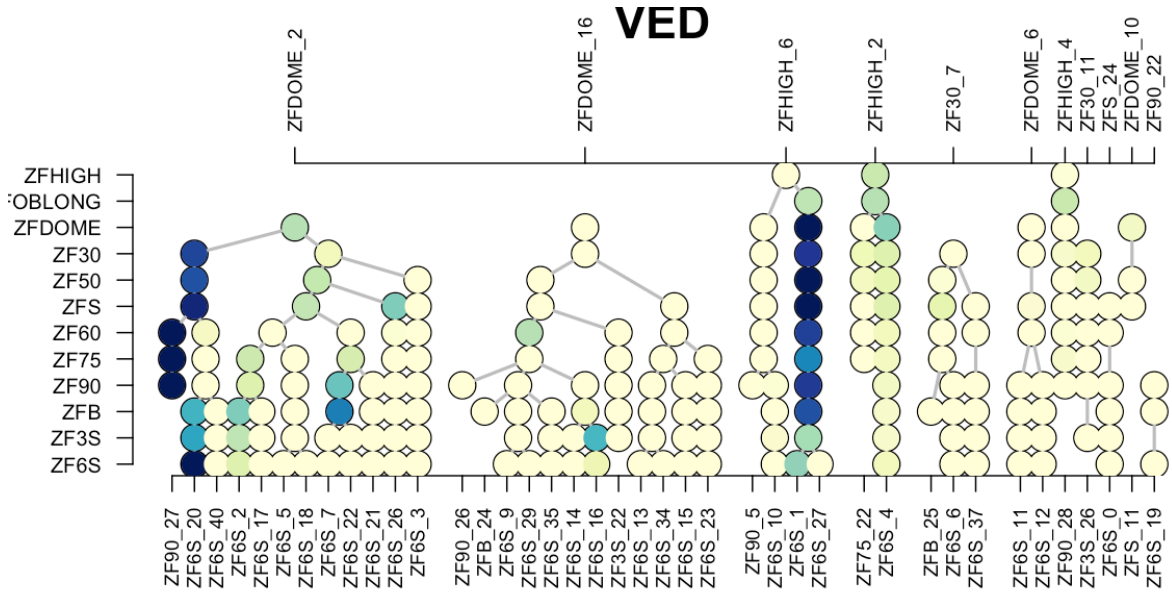
Notochord marker gene

```
plot_MTree("NOTO", M.tree)
```



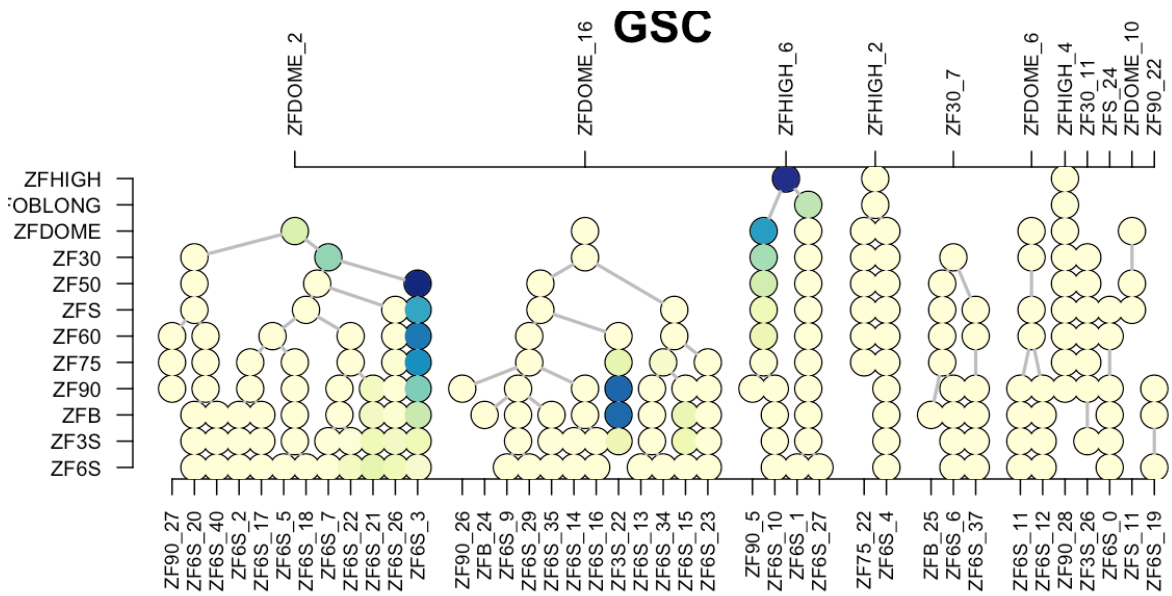
Ectoderm marker gene

```
plot_MTree("VED", M.tree)
```



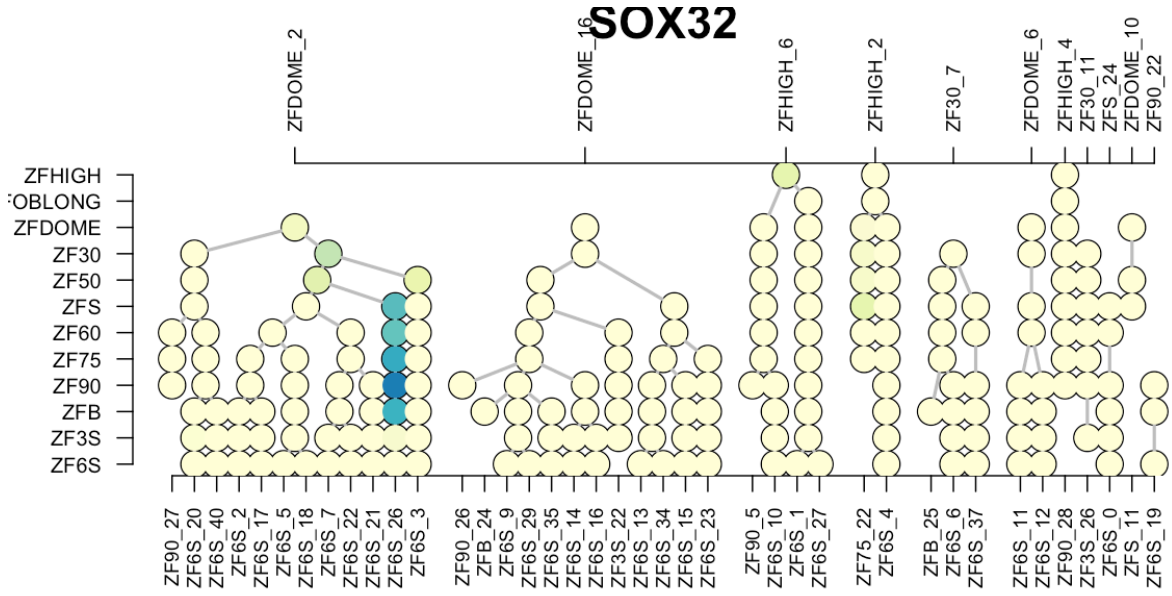
prechordal plate marker gene

```
plot_MTree("GSC", M.tree)
```



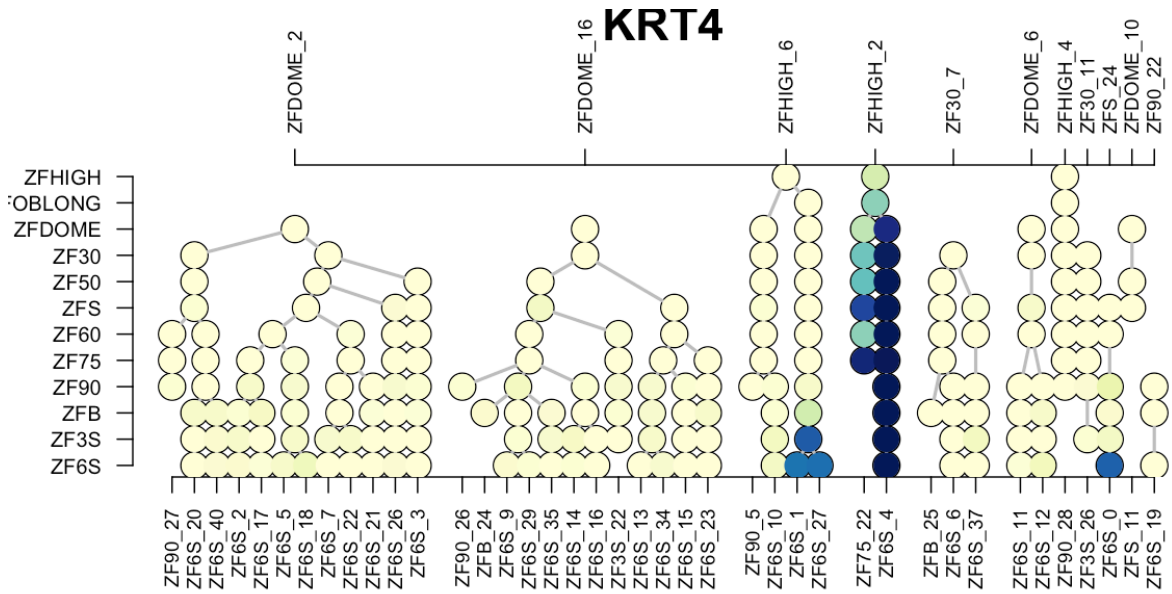
endoderm marker gene

```
plot_MTree("SOX32", M.tree)
```



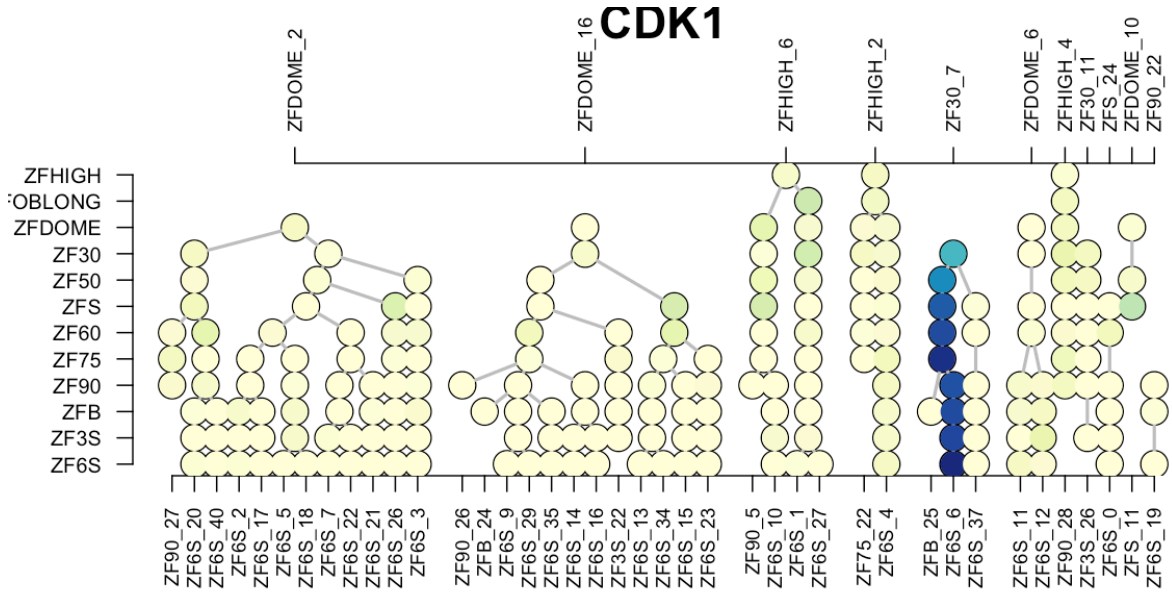
EVL marker gene

```
plot_MTree("KRT4", M.tree)
```



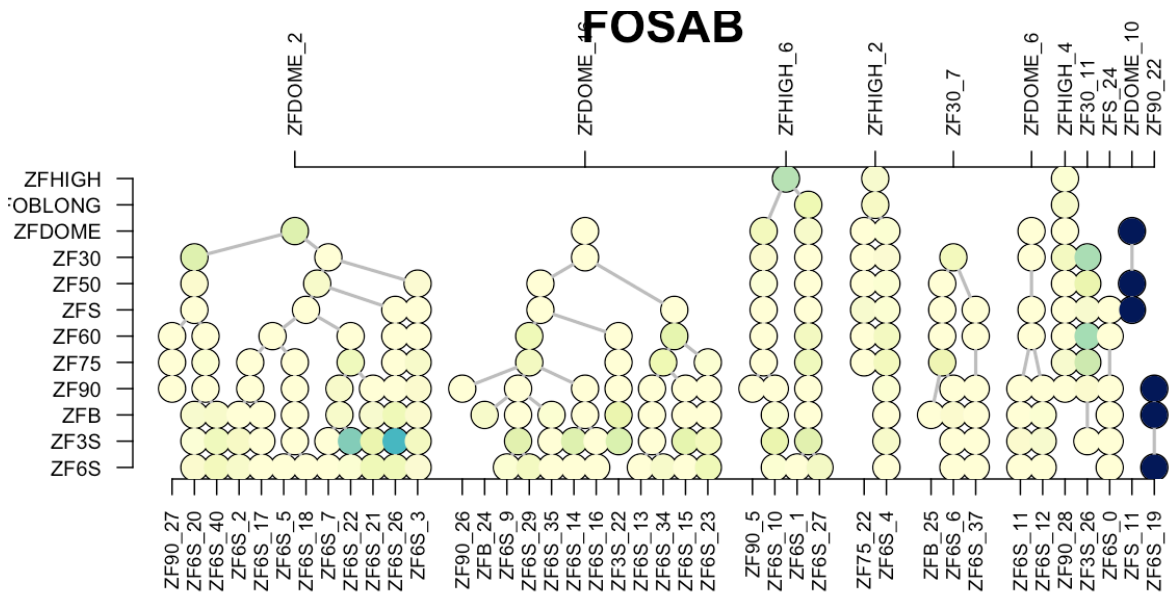
Cell cycle gene

```
plot_MTree("CDK1", M.tree)
```



Stress response gene

```
plot_MTree("FOSAB", M.tree)
```


FOSAB
ZFDOME_2
ZFDOME_46
ZFHIGH_6
ZFHIGH_2
ZF30_7
ZFDOME_6
ZFHIGH_4
ZF30_11
ZFS_24
ZFDOME_10
ZF90_22
ZFHIGH
OBLONG
ZFDOME
ZF30
ZF50
ZFS
ZF60
ZF75
ZF90
ZFB
ZF3S
ZF6S
ZF90_27
ZF6S_20
ZF6S_40
ZF6S_2
ZF6S_17
ZF6S_5
ZF6S_18
ZF6S_7
ZF6S_22
ZF6S_21
ZF6S_26
ZF6S_3
ZF90_26
ZFB_24
ZF6S_9
ZF6S_29
ZF6S_35
ZF6S_14
ZF6S_16
ZF3S_22
ZF6S_13
ZF6S_34
ZF6S_15
ZF6S_23
ZF90_5
ZF6S_10
ZF6S_1
ZF6S_27
ZF75_22
ZF6S_4
ZFB_25
ZF6S_6
ZF6S_37
ZF6S_11
ZF6S_12
ZF90_28
ZF3S_26
ZF6S_0
ZFS_11
ZF6S_19