SA09: Hydra URD Endoderm

Jeff Farrell

October 1, 2018

Setup

```
# Load required libraries
library(URD)
```

```
## Loading required package: ggplot2
## Warning: package 'ggplot2' was built under R version 3.4.4
## Loading required package: Matrix
## Warning: package 'Matrix' was built under R version 3.4.2
```

```
# Set main
opts_chunk$set(root.dir = "~/Dropbox/HydraURDSubmission/")
main.path <- "~/Dropbox/HydraURDSubmission/"

# Build output directory
dir.create(paste0(main.path, "URD/Endoderm/"), recursive = T)
```

```
## Warning in dir.create(paste0(main.path, "URD/Endoderm/"), recursive = T):
## '/Users/jaf2030/Dropbox/HydraURDSubmission/URD/Endoderm' already exists
```

Load Data

```
# Load subsetted endoderm Seurat object and convert to an URD object.
hydra.seurat <- readRDS(paste0(main.path, "objects/Hydra_Seurat_Endo.rds"))
hydra.en <- seuratToURD(hydra.seurat)
```

```
## [1] "Marchenko-Pastur eigenvalue null upper bound: 5.84495621200769"
## [1] "12 PCs have larger eigenvalues."
```

```
rm(hydra.seurat)

# Load full-data transcriptome Seurat object and cover to an URD object.
hydra.seurat <- readRDS(paste0(main.path, "objects/Hydra_Seurat_Whole_Transcriptome.rds"))
hydra.full <- seuratToURD(hydra.seurat)
```

```
## [1] "Marchenko-Pastur eigenvalue null upper bound: 18.6992548391687"
## [1] "11 PCs have larger eigenvalues."
```

```
rm(hydra.seurat)

# Load NMF results
nmf.full.cells <- as(as.matrix(t(read.csv(paste0(main.path, "nmf/wt_K96/GoodMeta_CellScores.csv"),
    row.names = 1))), "dgCMatrix")
nmf.full.genes <- as(as.matrix(read.csv(paste0(main.path, "nmf/wt_K96/GoodMeta_GeneScores.csv"),
    row.names = 1)), "dgCMatrix")
nmf.endo.cells <- as(as.matrix(t(read.csv(paste0(main.path, "nmf/en_K40/GoodMeta_CellScores.csv"),
    row.names = 1))), "dgCMatrix")
nmf.endo2.cells <- as(as.matrix(t(read.csv(paste0(main.path, "nmf/en_K40/BadMeta_CellScores.csv"),
    row.names = 1))), "dgCMatrix")
rownames(nmf.full.cells) <- gsub("X", "", gsub("\\.", "-", rownames(nmf.full.cells)))
rownames(nmf.endo.cells) <- gsub("X", "", gsub("\\.", "-", rownames(nmf.endo.cells)))
rownames(nmf.endo2.cells) <- gsub("X", "", gsub("\\.", "-", rownames(nmf.endo.cells)))

# Scale NMF results to 0-1
nmf.full.cells <- sweep(nmf.full.cells, 2, apply(nmf.full.cells, 2, max), "/")

# Put NMF results into URD objects
hydra.en@nmf.c1 <- cbind(nmf.full.cells[colnames(hydra.en@logupx.data), ], nmf.endo.cells[colnames(hydra.en@logupx.data),
    ], nmf.endo2.cells[colnames(hydra.en@logupx.data), ])
hydra.full@nmf.c1 <- nmf.full.cells[colnames(hydra.full@logupx.data), ]

# Keep track of NMF modules for later
nmf.mods.en <- c(colnames(nmf.endo.cells), colnames(nmf.endo2.cells))
nmf.mods.f <- colnames(nmf.full.cells)
```

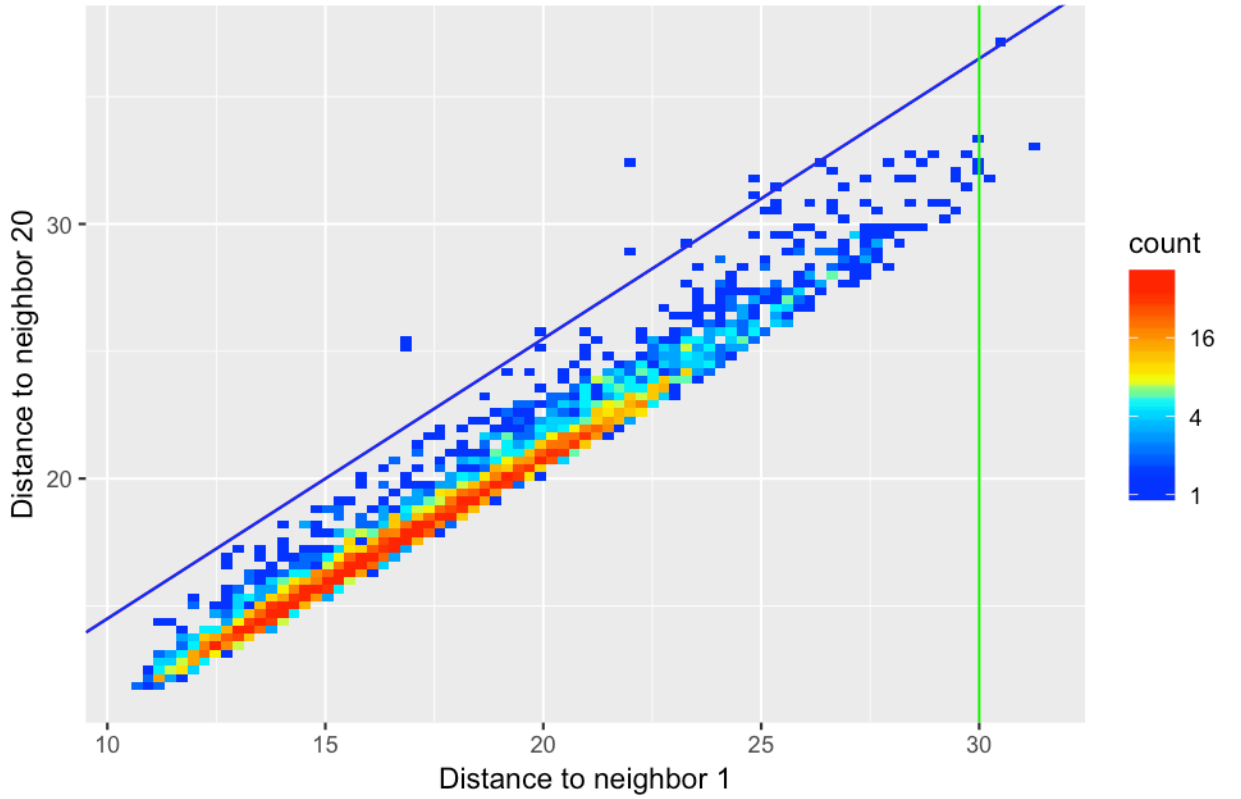
Remove outlier cells and doublets

Before attempting to build trajectories, it's important to clean the data by removing outlier cells that will confound the calculation of transition probabilities and removing doublets. Due to the phagocytic behavior of the dissociated endodermal cells, many cells that exhibit non-endodermal gene expression have to be removed.

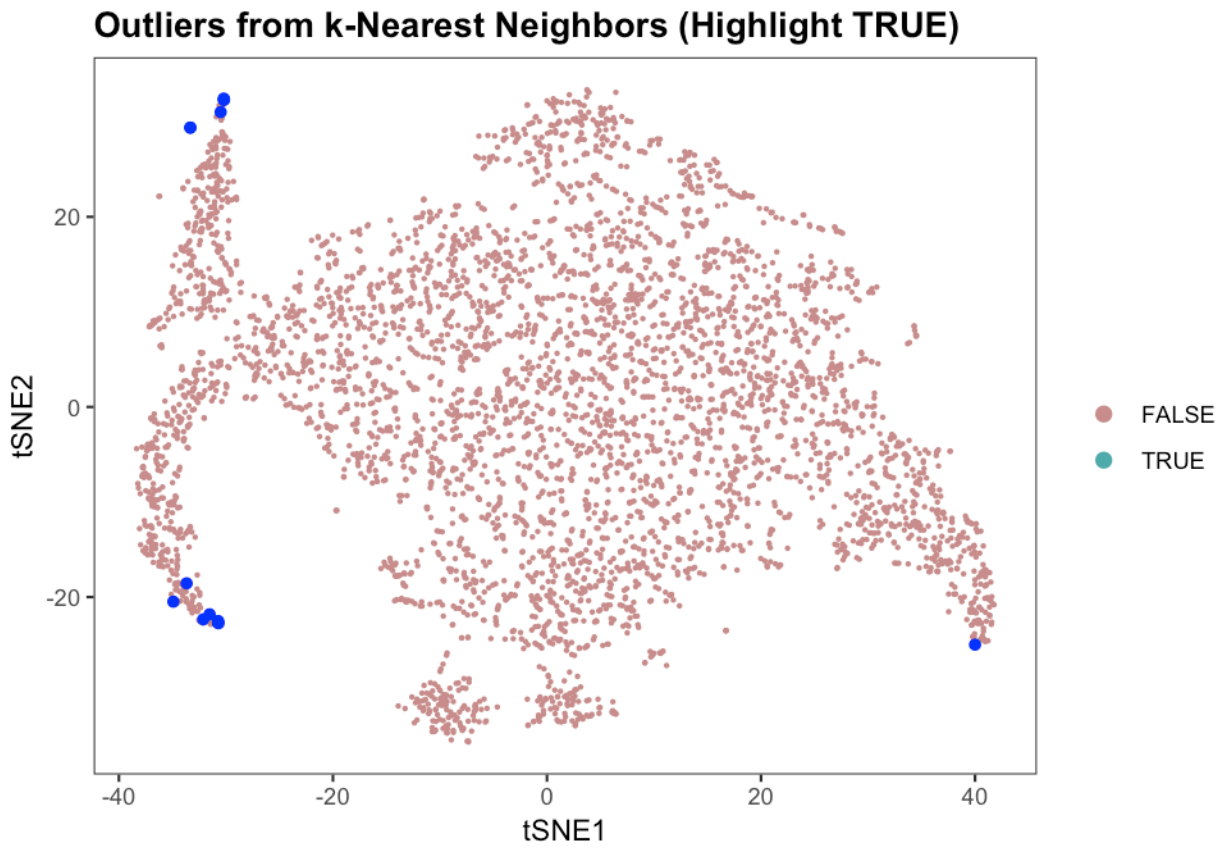
Calculate k-nearest neighbor networks

A few cells have unusually large distances to their nearest neighbors and will perform poorly in the diffusion map, so it is good to remove those.

```
# Calculate k-nearest neighbor network
hydra.en <- calcKNN(hydra.en, nn = 200)

# Choose outliers that have very unusual nearest neighbor distances
outliers.knn <- knnOutliers(hydra.en, x.max = 30, slope.r = 1.1, int.r = 3.5, slope.b = 1.1,
    int.b = 3.5)
```



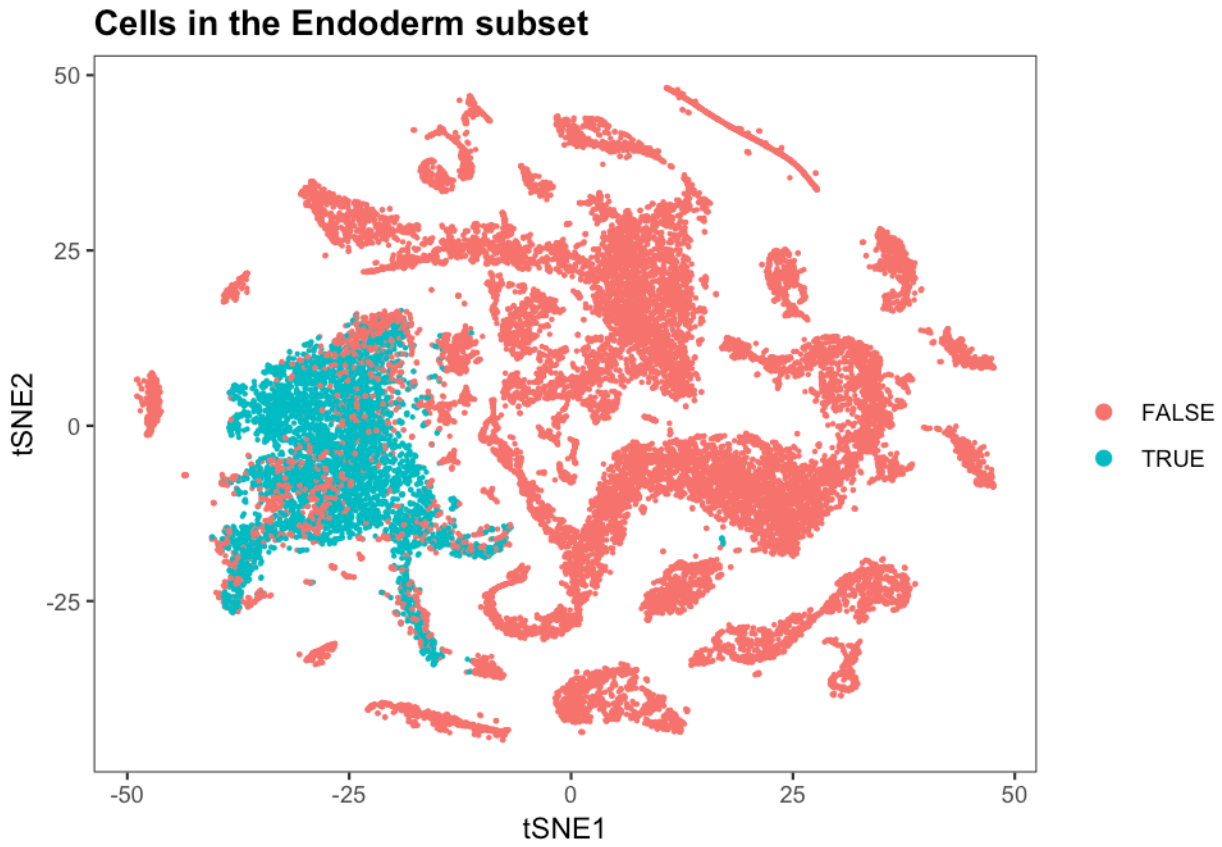

```
# Check out who those outliers are to make sure you're not cropping out all
# differentiated cells or something
hydra.en <- groupFromCells(hydra.en, group.id = "knn.outliers", cells = outliers.knn)
plotDimHighlight(hydra.en, "knn.outliers", "TRUE", highlight.color = "blue", plot.title = "Outliers from k-Nearest Neighbors")
```



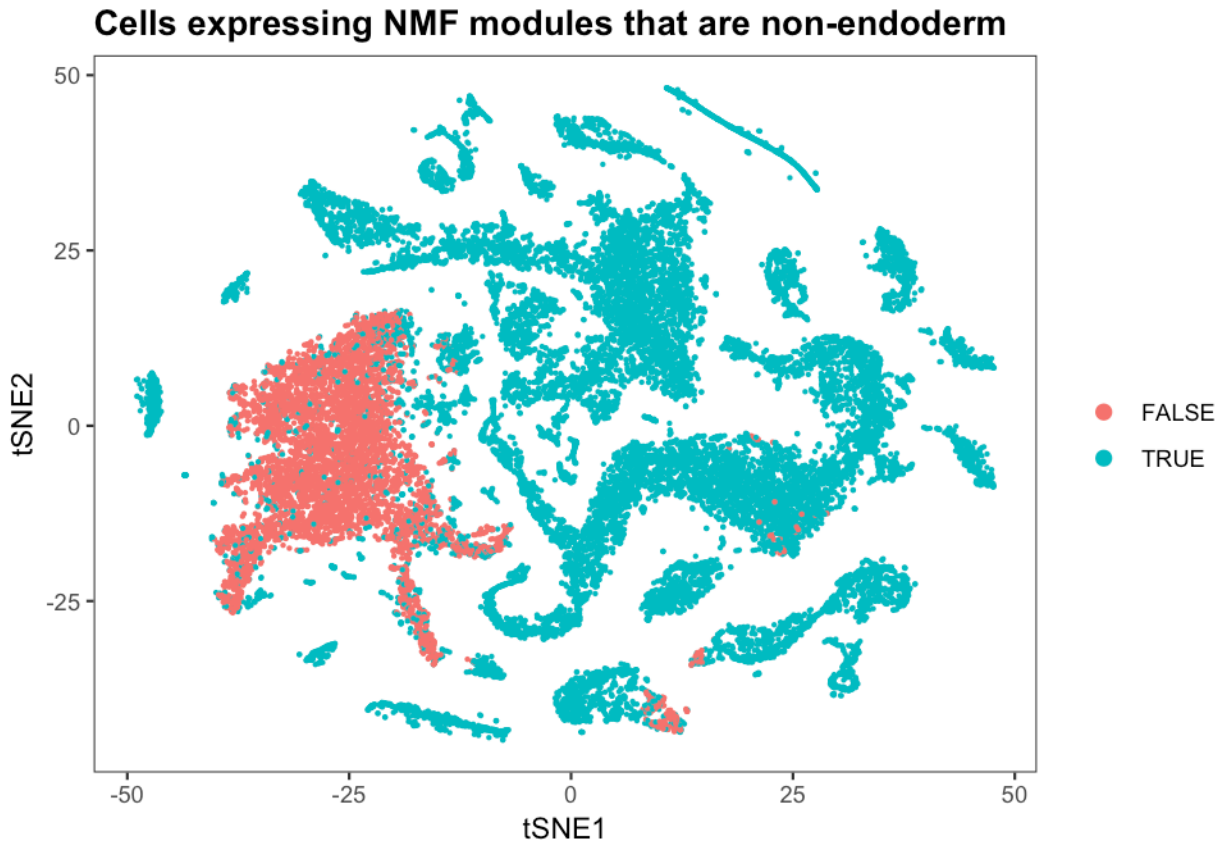
Non-endodermal contaminating cells

Many cells in the data represent biological doublets, since cells of the endodermal lineage readily engulf other cells once dissociated. In order to remove these cells, we used a non-negative matrix factorization (NMF) decomposition of the full data set, and identified those modules that are not primarily expressed in the endoderm. Any cells in the 'endoderm' lineage that had significant expression of modules primarily associated with other cell types were removed.

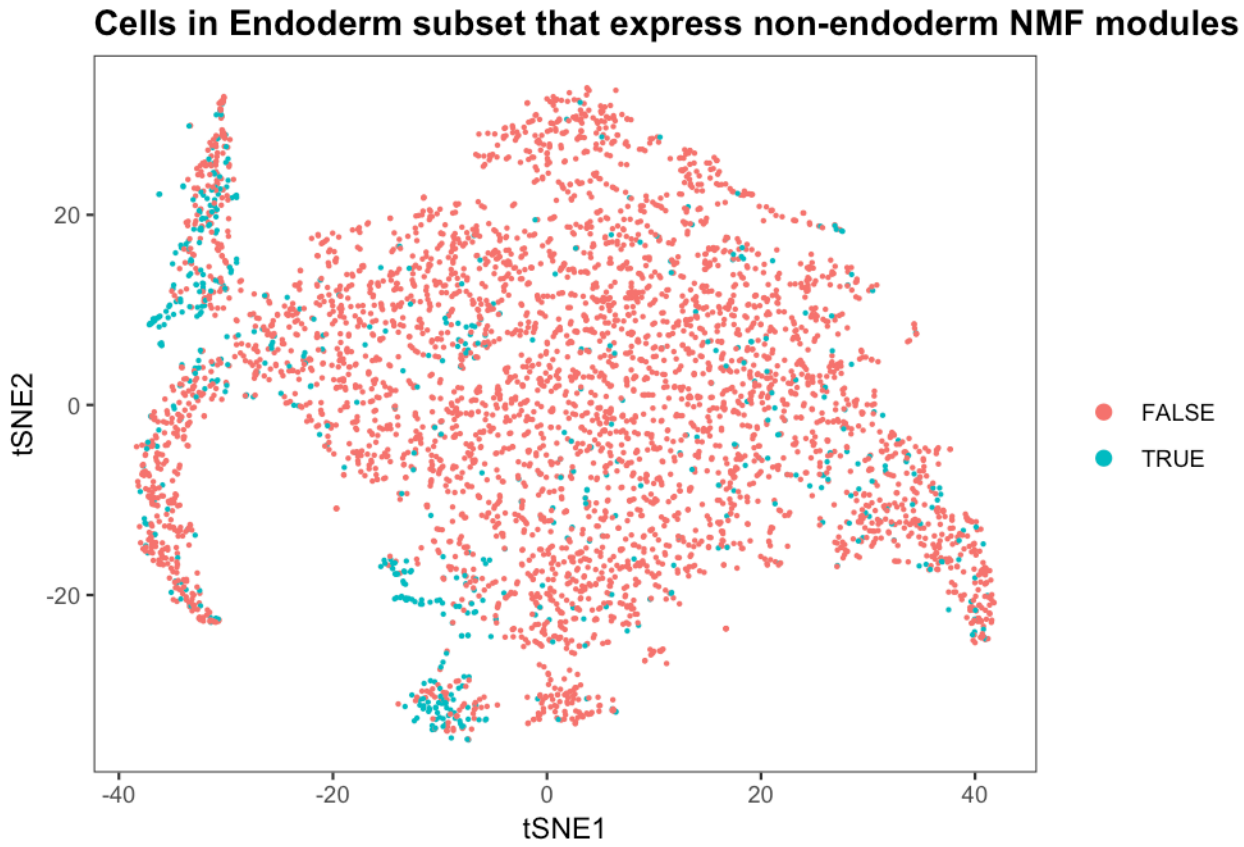
```
# Modules from NMF calculated on entire data that are representative of
# non-endodermal cells.
nmf.full.nonendo <- c("wt4", "wt7", "wt8", "wt9", "wt10", "wt11", "wt13", "wt15",
    "wt16", "wt18", "wt19", "wt20", "wt22", "wt23", "wt25", "wt32", "wt34", "wt36",
    "wt37", "wt39", "wt40", "wt41", "wt42", "wt43", "wt46", "wt47", "wt48", "wt49",
    "wt50", "wt53", "wt54", "wt55", "wt57", "wt61", "wt62", "wt63", "wt64", "wt66",
    "wt67", "wt69", "wt71", "wt72", "wt73", "wt74", "wt75", "wt76", "wt79", "wt80",
    "wt81", "wt85", "wt93", "wt33", "wt45", "wt65")

# Identify the cells in the full-data object that are in the endoderm object
hydra.full <- groupFromCells(hydra.full, "endoderm", colnames(hydra.en@logupx.data))
plotDim(hydra.full, "endoderm", plot.title = "Cells in the Endoderm subset")
```




```
# Which cells express NMF modules that are not associated with the endoderm?
# Here, expression is defined cells that express one or more modules at a level >
# 0.125 (scaled from 0-1)
cells.full.nmfnonendo <- names(which(Matrix::rowSums(hydra.full@nmf.c1[, nmf.full.nonendo] >
    0.125) > 0))
hydra.full <- groupFromCells(hydra.full, "nonendo", cells.full.nmfnonendo)
plotDim(hydra.full, "nonendo", plot.title = "Cells expressing NMF modules that are non-endoderm")
```



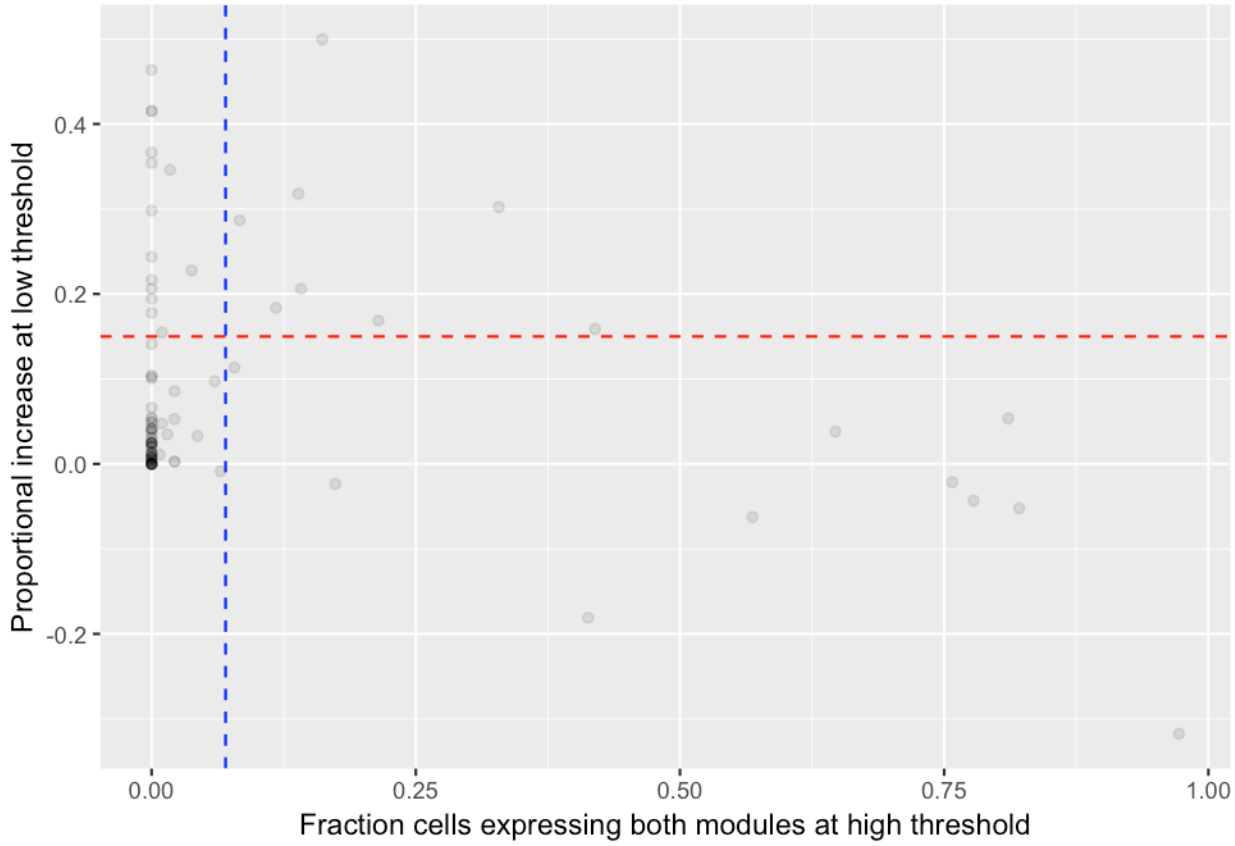

```
# Annotate the endoderm subset with cells that express non-endoderm NMF modules
outliers.nmf.full.nonendo <- intersect(cells.full.nmfnonendo, colnames(hydra.en@logupx.data))
hydra.en <- groupFromCells(hydra.en, "outliers.nmf.full.nonendo", outliers.nmf.full.nonendo)
plotDim(hydra.en, "outliers.nmf.full.nonendo", plot.title = "Cells in Endoderm subset that express non-endoderm NMF modules")
```



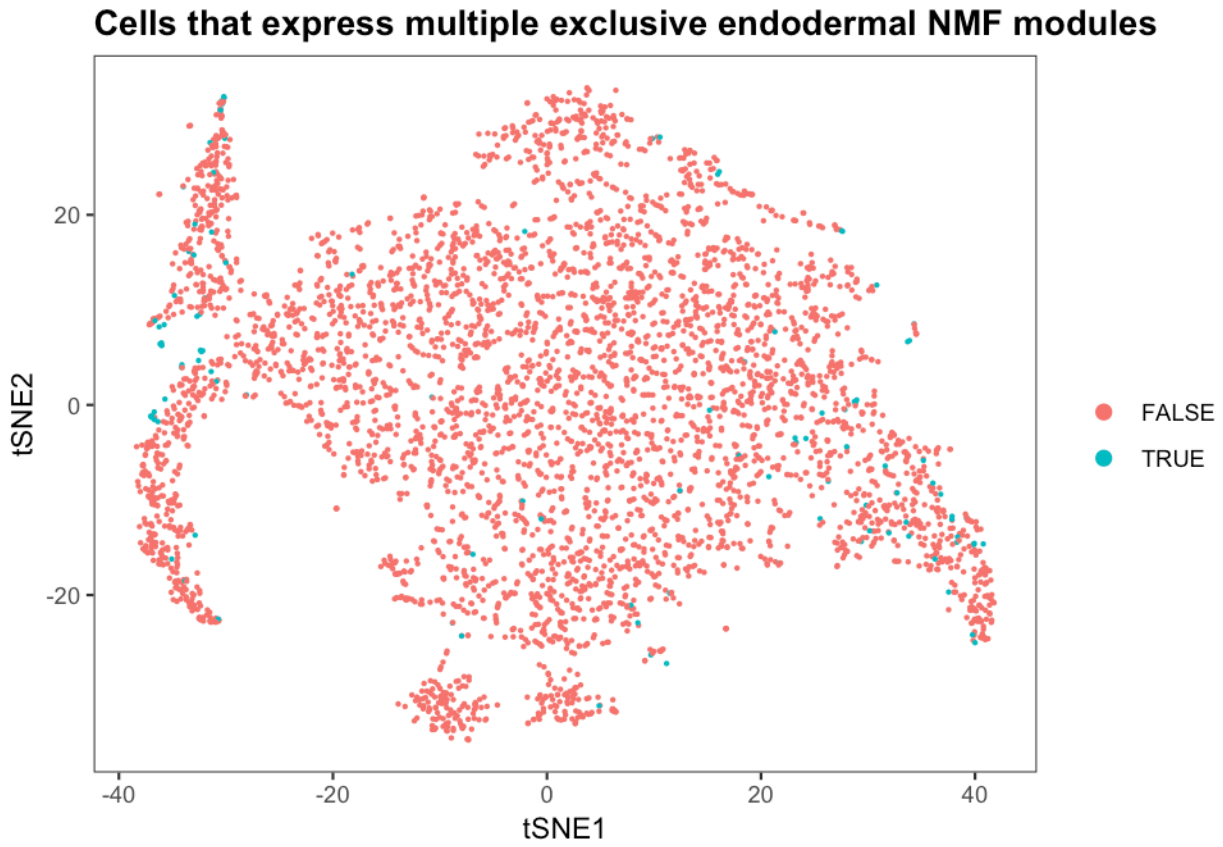
Doublets via NMF

Additionally, some endodermal cells are doublets of multiple endodermal cell types. These are detected using NMF decomposition of the endoderm lineage. These doublets are cells that express multiple NMF modules that are non-overlapping in the data. (In other words, NMF modules that are mutually exclusive in the majority of the data.)

```
nmf.en.mods.use <- c("en6", "en7", "en8", "en9", "en13", "en15", "en21", "en23",
    "en25", "en26", "en31", "en32", "en35", "en39")

# Determine overlaps between module pairs
nmf.doublet.combos <- NMFDoubletsDefineModules(hydra.en, modules.use = nmf.en.mods.use,
    module.thresh.high = 0.3, module.thresh.low = 0.15)

# Determine which module pairs to use for doublets
NMFDoubletsPlotModuleThresholds(nmf.doublet.combos, frac.overlap.max = 0.07, frac.overlap.diff.max = 0.15)
```

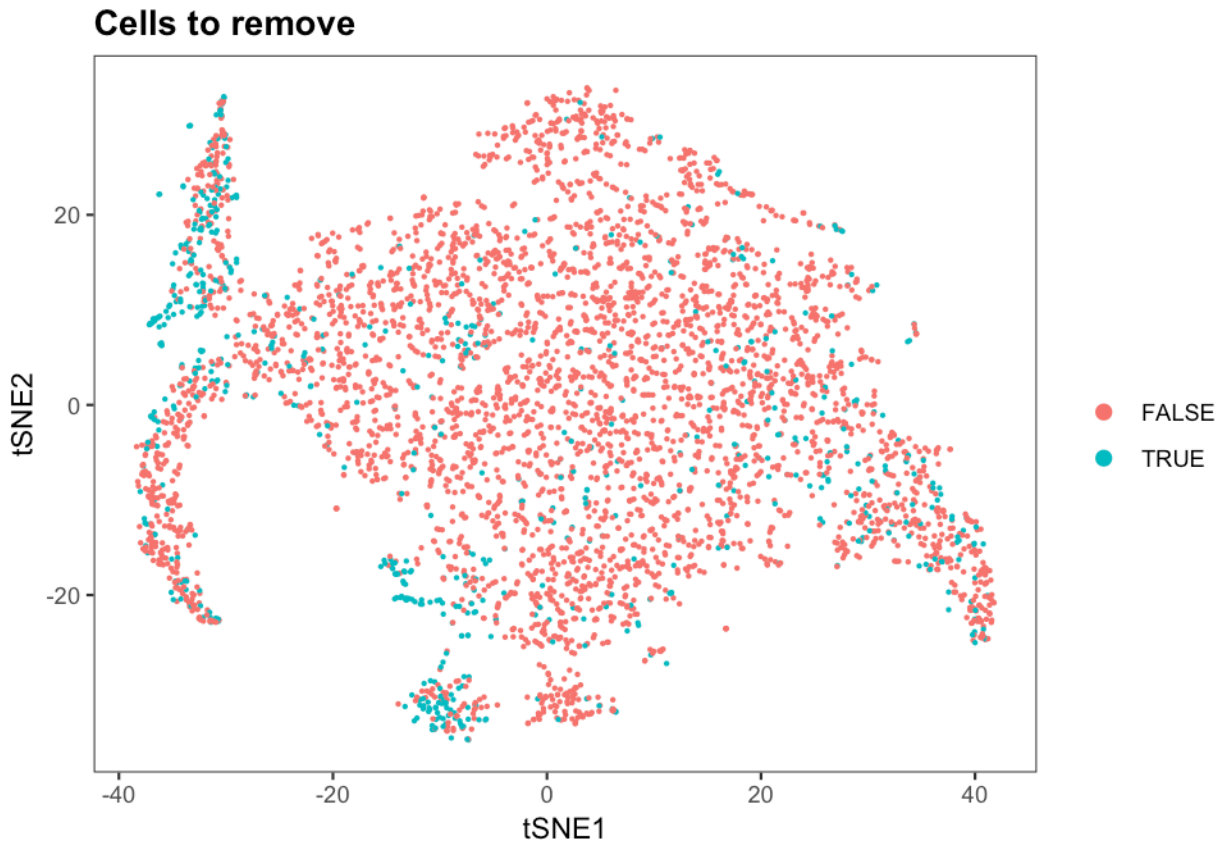



```
# Define doublet cells
nmf.doublets <- NMFDoubletsDetermineCells(hydra.en, nmf.doublet.combos, module.expressed.thresh = 0.2,
    frac.overlap.max = 0.07, frac.overlap.diff.max = 0.15)  # 114 cells

# Visualize doublets
hydra.en <- groupFromCells(hydra.en, "nmf.doublets", nmf.doublets)
plotDim(hydra.en, "nmf.doublets", plot.title = "Cells that express multiple exclusive endodermal NMF modules")
```



Generate subsetted data

```
# All cropped cells
cropped.cells <- unique(c(outliers.knn, outliers.nmf.full.nonendo, nmf.doublets))
hydra.en <- groupFromCells(hydra.en, "cropped.cells", cropped.cells)
plotDim(hydra.en, "cropped.cells", plot.title = "Cells to remove")
```



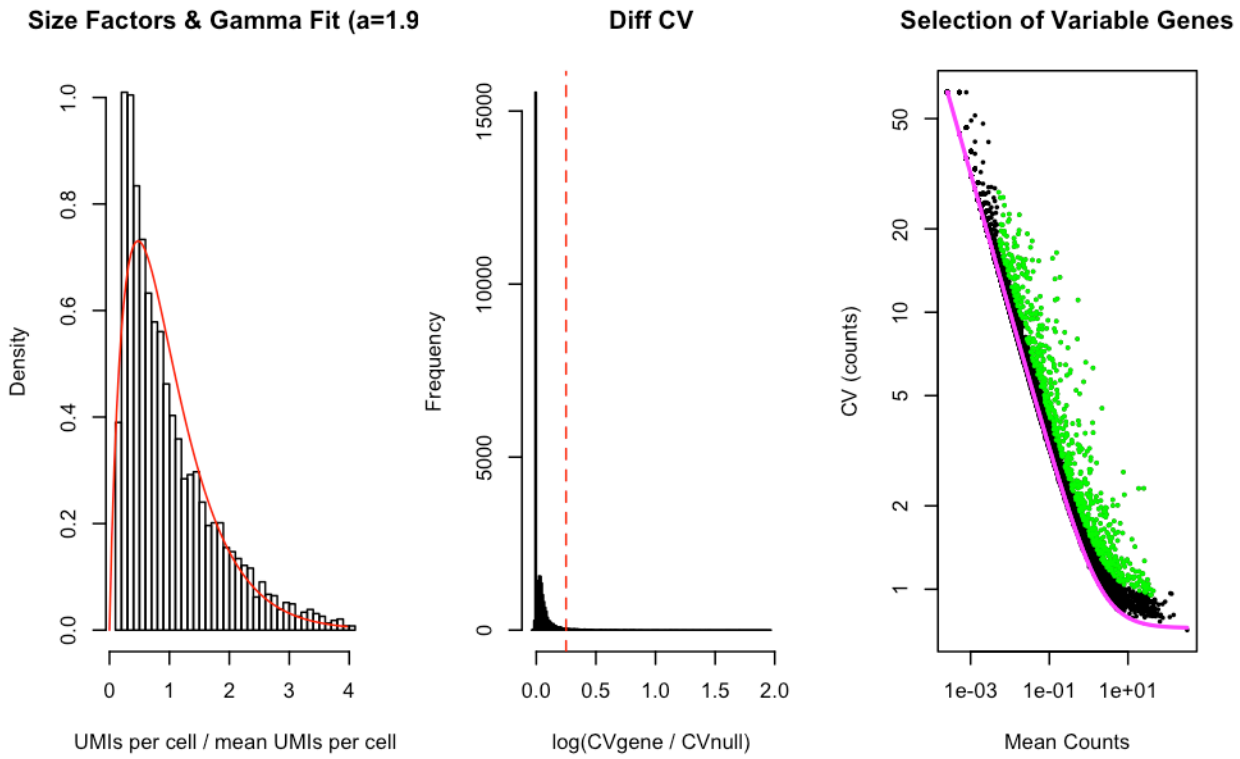

```
# Define new cell populations
cells.no.outliers <- setdiff(colnames(hydra.en@logupx.data), cropped.cells)

# Subset the URD object
hydra.en <- urdSubset(hydra.en, cells.no.outliers)
```

Variable genes

We perform the analysis on genes that exhibit more variability than other genes of similar expression levels in order to focus on the genes that are most likely to encode biological information (as opposed to just technical). Since we've subsetted the object and also removed lots of contaminating cells, we should re-calculate the variable genes to focus just on the endoderm.

```
# A more restrictive list of variable genes
new.var <- findVariableGenes(object = hydra.en, set.object.var.genes = F, diffCV.cutoff = 0.25,
    do.plot = T)  # 719 genes
```

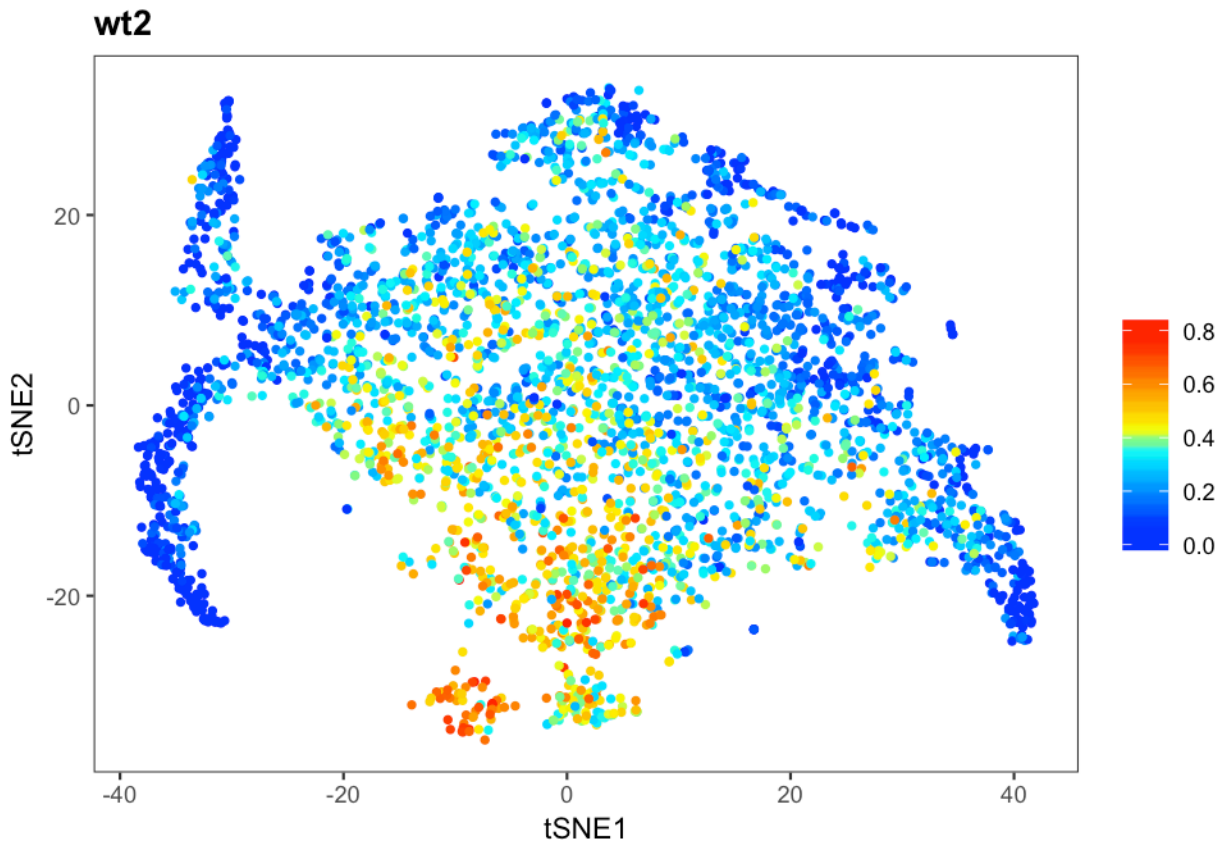
```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 4704 x values <= 0 omitted
## from logarithmic plot
```



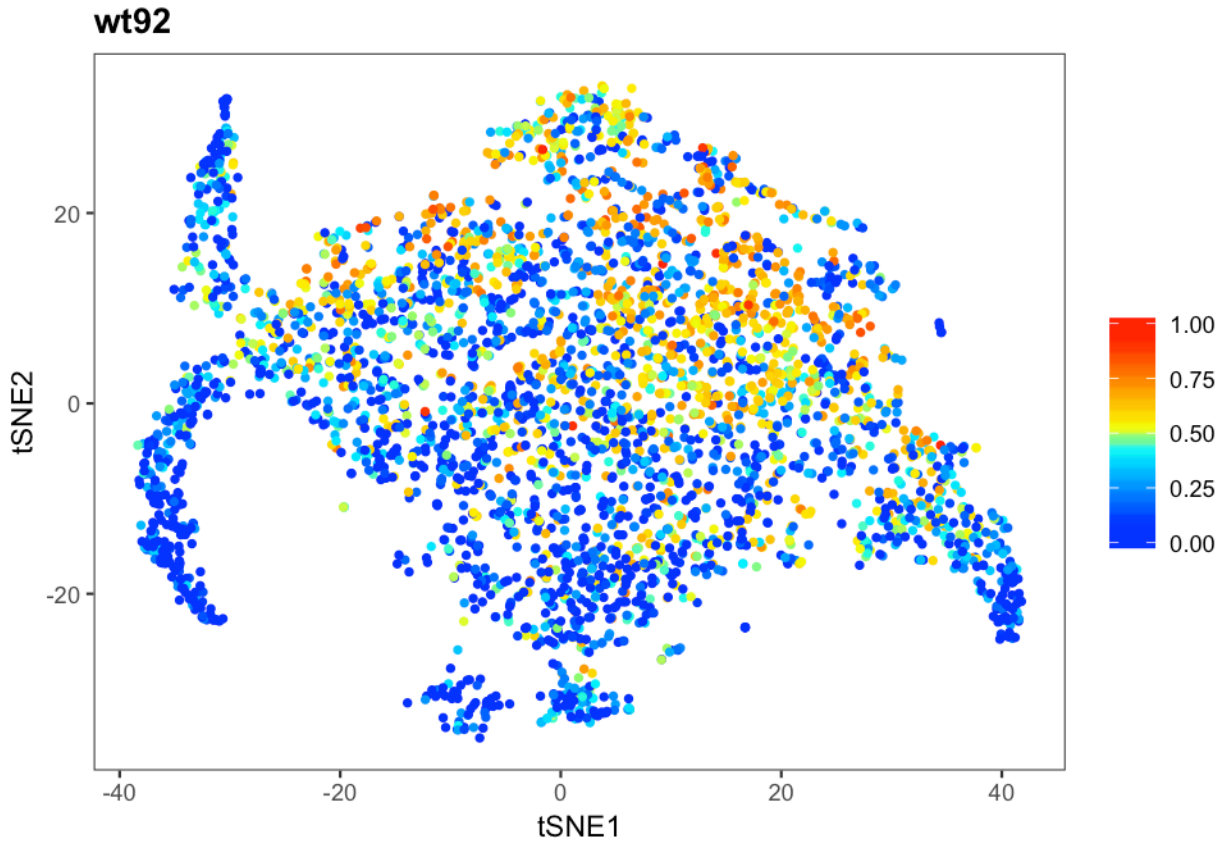
Stress-axis genes

In the tSNE, it was evident that there was a strong signal in the body column separate from the spatial trajectory. Further investigation revealed two NMF modules that were strongly expressed along that dimension (wt2 and wt92) that seem to encode a stress module (wt2) and an anti-stress module (wt92). We removed the genes that are strongly loaded in those NMF modules from the variable genes in order to focus on the spatial trajectory of interest, rather than the putative stress response.

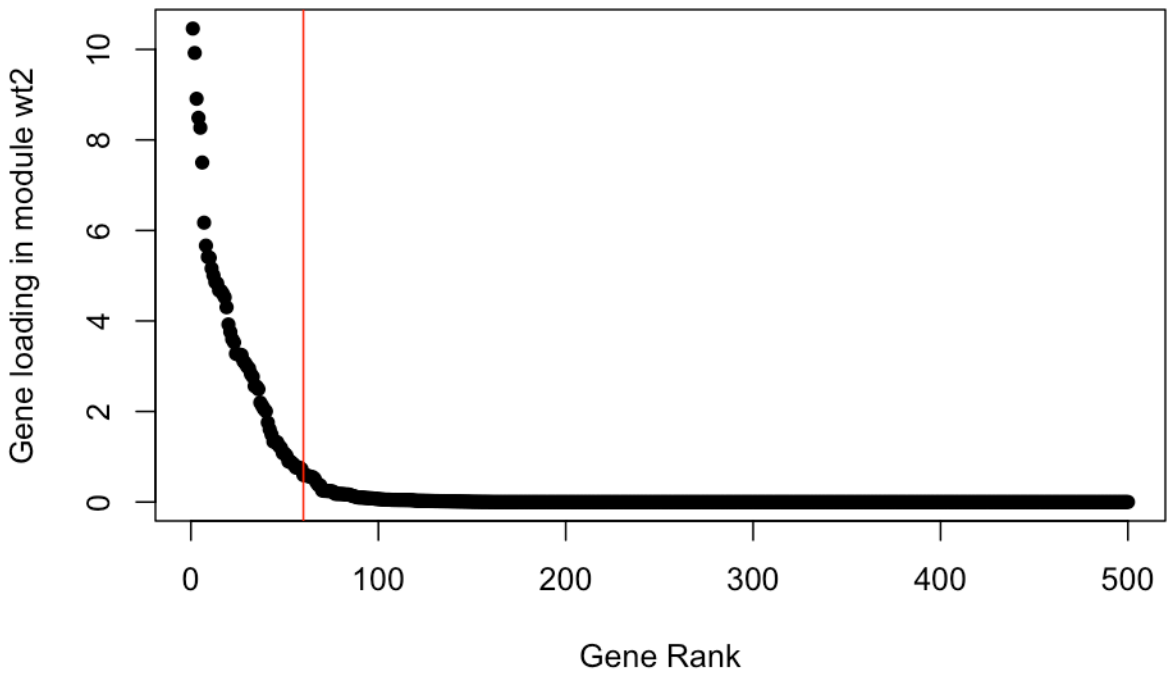
```
plotDim(hydra.en, "wt2")
```



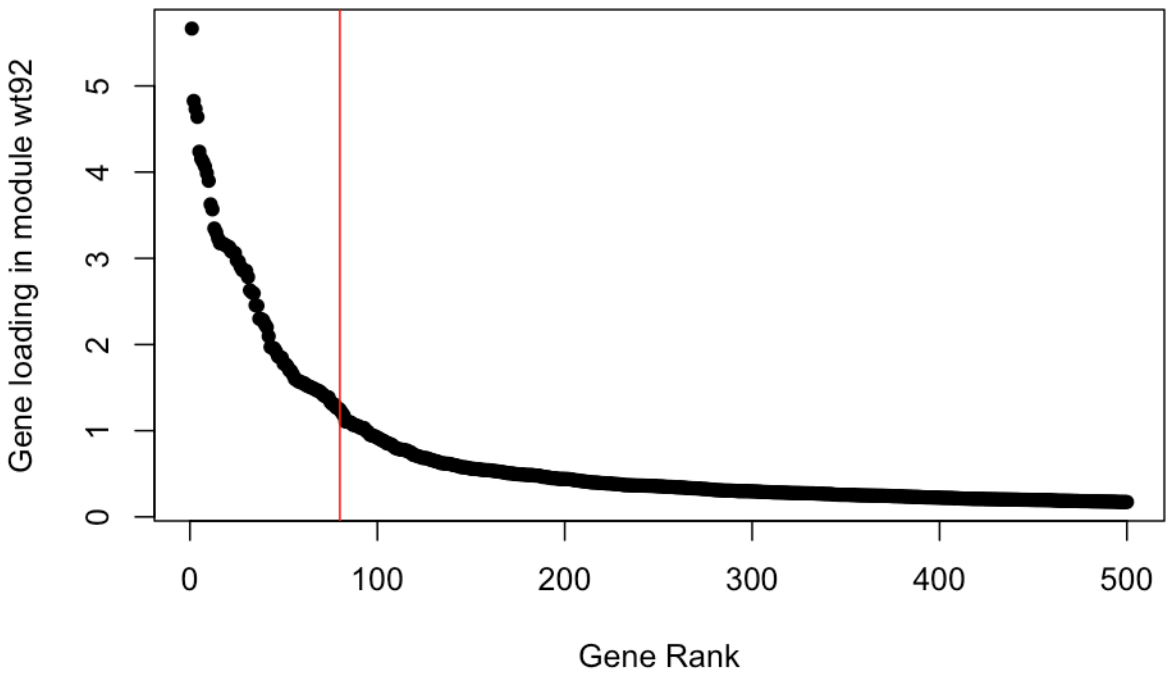

```
plotDim(hydra.en, "wt92")
```




```
# Look at the loadings of genes into NMF modules wt2, and choose a cut-off near
# the elbow
wt2.top500 <- order(nmf.full.genes[, "wt2"], decreasing = T)[1:500]
plot(x = 1:500, y = nmf.full.genes[wt2.top500, "wt2"], pch = 16, xlab = "Gene Rank",
    ylab = "Gene loading in module wt2")
abline(v = 60, col = "red")
```




```
wt2.genes <- rownames(nmf.full.genes)[wt2.top500][1:60]

# Look at the loadings of genes into NMF modules wt92, and choose a cut-off near
# the elbow
wt92.top500 <- order(nmf.full.genes[, "wt92"], decreasing = T)[1:500]
plot(x = 1:500, y = nmf.full.genes[wt92.top500, "wt92"], pch = 16, xlab = "Gene Rank",
    ylab = "Gene loading in module wt92")
abline(v = 80, col = "red")
```




```
wt92.genes <- rownames(nmf.full.genes)[wt92.top500][1:80]

# Remove genes that were strongly loaded into the stress-responsive modules
new.var <- setdiff(new.var, c(wt2.genes, wt92.genes))  # 674 genes, 45 stress-responsive genes were removed
```

Graph Clustering

Generated several more fine-grained clusterings in order to have better options for choosing root and tip cells.

```
# Set random seed so clusters get the same names every time
set.seed(19)
# Graph clustering with various parameters
hydra.en <- graphClustering(hydra.en, num.nn = c(20, 30, 40, 50), do.jaccard = T,
    method = "Infomap")
hydra.en <- graphClustering(hydra.en, num.nn = c(5, 10, 15, 20, 30), do.jaccard = T,
    method = "Louvain")
hydra.en <- graphClustering(hydra.en, num.nn = c(6, 7, 8), do.jaccard = T, method = "Louvain")
# Record the clustering
pdf(paste0(main.path, "URD/Endoderm/Clusters-Endoderm-Infomap30.pdf"))
plotDim(hydra.en, "Infomap-30", label.clusters = T, legend = F)
dev.off()
```

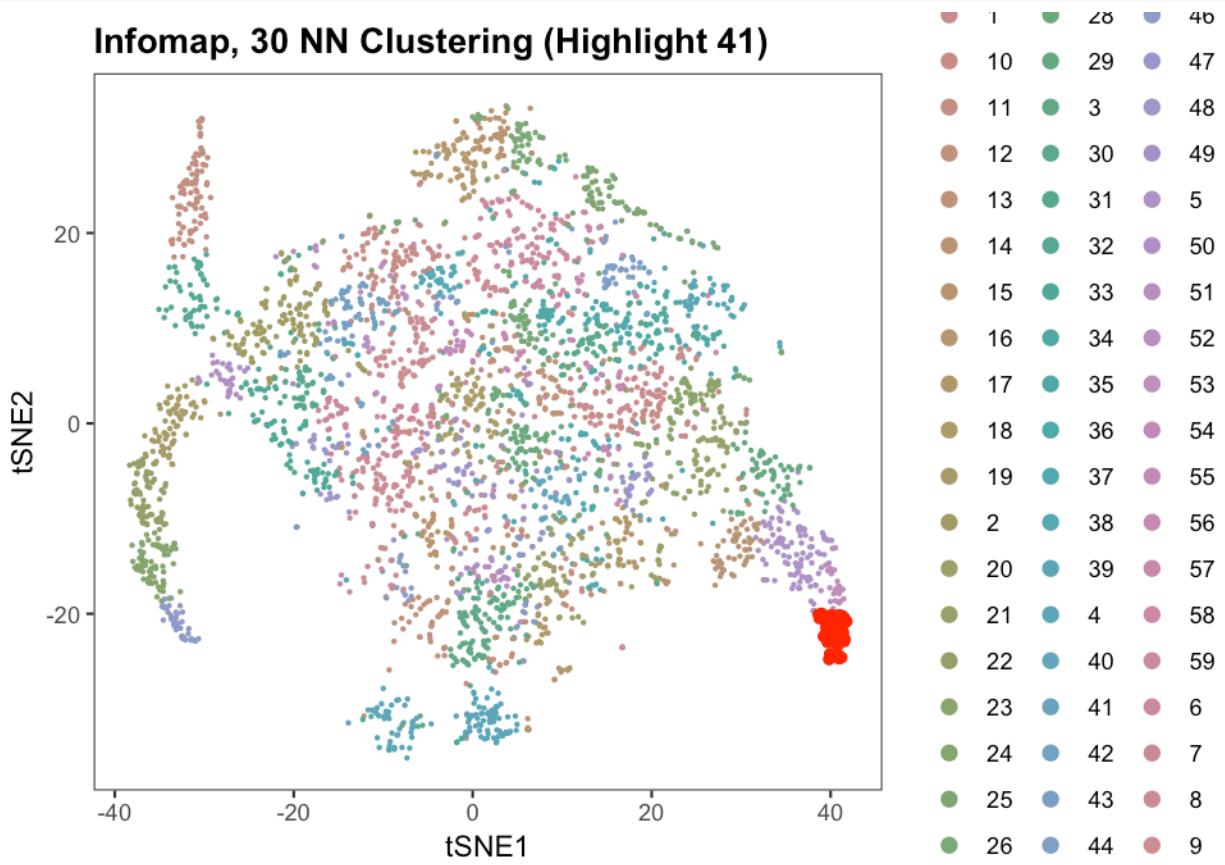
```
## quartz_off_screen
##                 2
```

Define root and tips

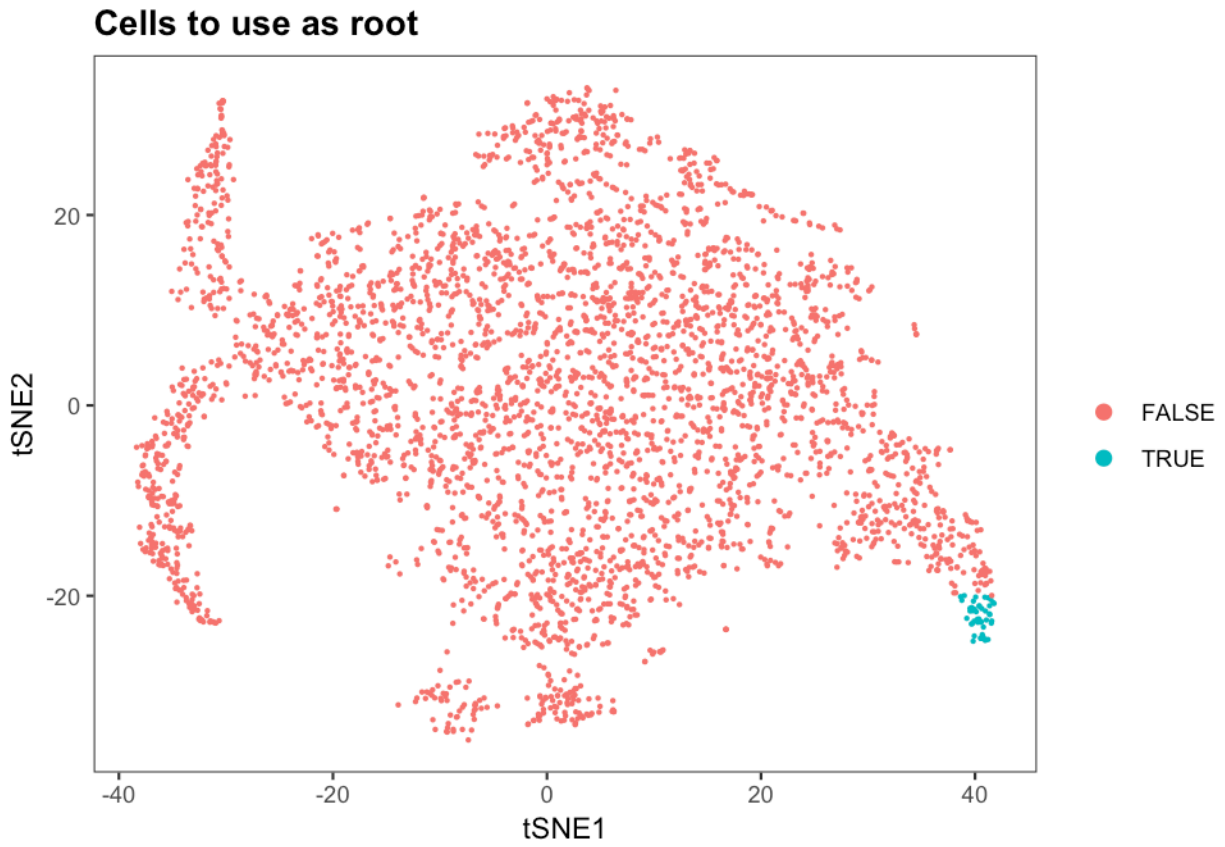
Root

Here, we are building spatial trajectories in the endoderm, so going to use the foot (cluster 41) as the root and then head towards the hypostome and tentacles to build a braching trajectory.

```
# Using Infomap, 30 NN clustering to define roots and tips
plotDimHighlight(hydra.en, clustering = "Infomap-30", cluster = "41", plot.title = "Infomap, 30 NN Clustering")
```



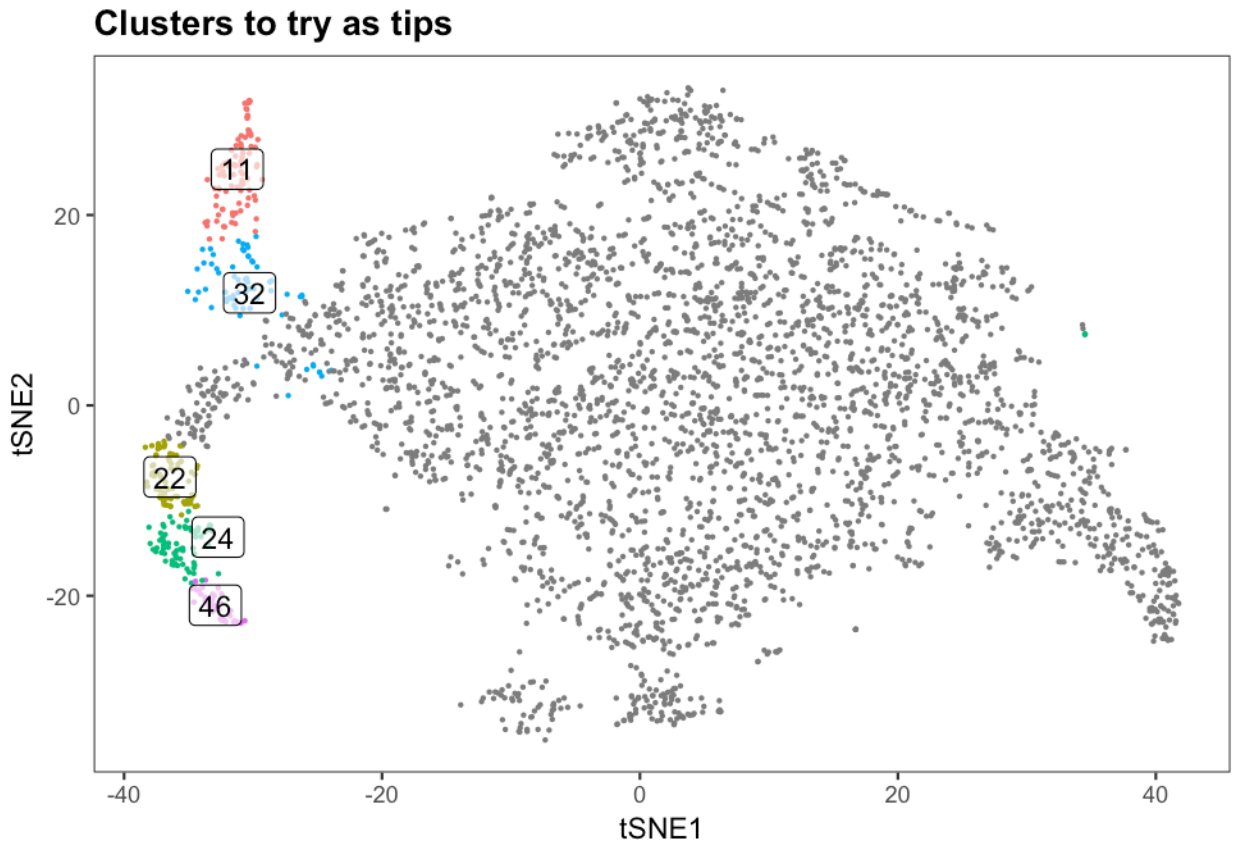

```
# Define root cells
root.cells <- cellsInCluster(hydra.en, "Infomap-30", "41")
hydra.en <- groupFromCells(hydra.en, "root", root.cells)

# Record info about them.
plotDim(hydra.en, "root", plot.title = "Cells to use as root")
```



Tips

Going to try a few clusters each from the hypostome and tentacle as starting points for the random walk. Occasionally, the most terminal cluster is poorly connected and doesn't do a good job of walking through the rest of the data.

```
# Establish a tip clustering
hydra.en@group.ids$tip.clusters <- NA
infomap30.clusters.use <- c("11", "32", "46", "24", "22")
i30.tip.cells <- cellsInCluster(hydra.en, "Infomap-30", infomap30.clusters.use)
hydra.en@group.ids[i30.tip.cells, "tip.clusters"] <- hydra.en@group.ids[i30.tip.cells,
    "Infomap-30"]

plotDim(hydra.en, "tip.clusters", label.clusters = T, legend = F, plot.title = "Clusters to try as tips")
```



Diffusion maps

Calculate the transition probabilities

We calculated our diffusion map using 60 nearest neighbors (~ square root of number of cells in data), excluding two small contaminant/stress clusters, and using the restrictive variable gene list with stress genes removed. We used a global sigma that is slightly smaller than the 'autodetected' one (destiny suggests 8; here, we used 6.)

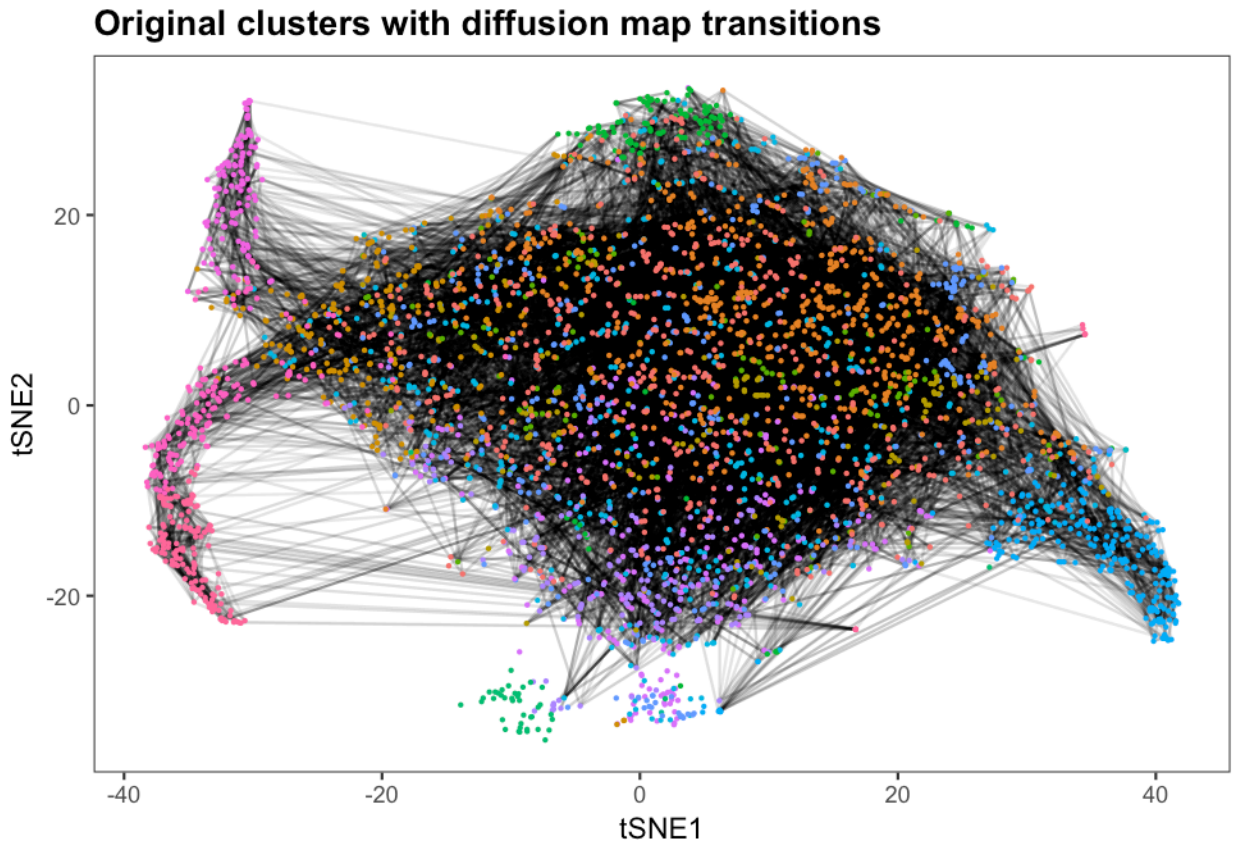
```
# Define cells without the contamination/stress clusters There are also these two
# clusters at the bottom that seems to be largely stress or contaminant clusters.
cells.contam.clusters <- cellsInCluster(hydra.en, "res.1.2", c("10", "12"))
cells.no.outliers.contam <- setdiff(colnames(hydra.en@logupx.data), cells.contam.clusters)

# Calculate diffusion map 60 NN = ~sqrt(3749 cells) Global detected sigma = 8,
# using sigma 6.
hydra.en <- calcDM(hydra.en, genes.use = new.var, cells.use = cells.no.outliers.contam,
    knn = 60, sigma.use = 6)
```

```
## [1] "Using provided global sigma 6"
```

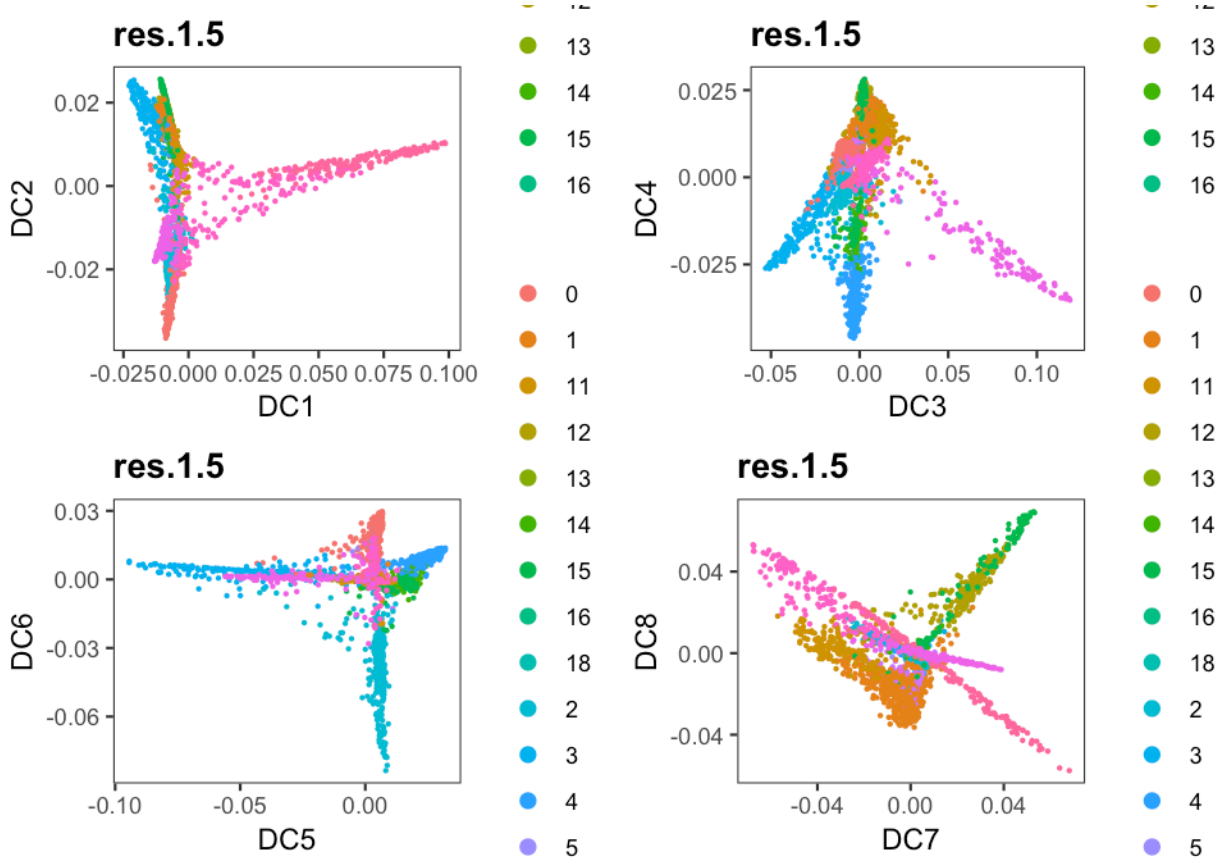
Evaluation of diffusion map

Diffusion map parameters seem good in that tentacle, hypostome, and foot are all nicely represented as spikes in the diffusion map and the connections in the diffusion map seem to do a good job of evenly connecting cells in the body column.

```
# Test how the connections are in the data
plotDim(hydra.en, "res.1.5", transitions.plot = 10000, plot.title = "Original clusters with diffusion map transitions",
    legend = F)
```




```
# These look good -- everything is well connected

# The hypostome, tentacle, and foot are well-represented as tips in the diffusion
# map, so parameters are probably fairly good.
plotDimArray(hydra.en, label = "res.1.5", reduction.use = "dm", dims.to.plot = 1:8)
```



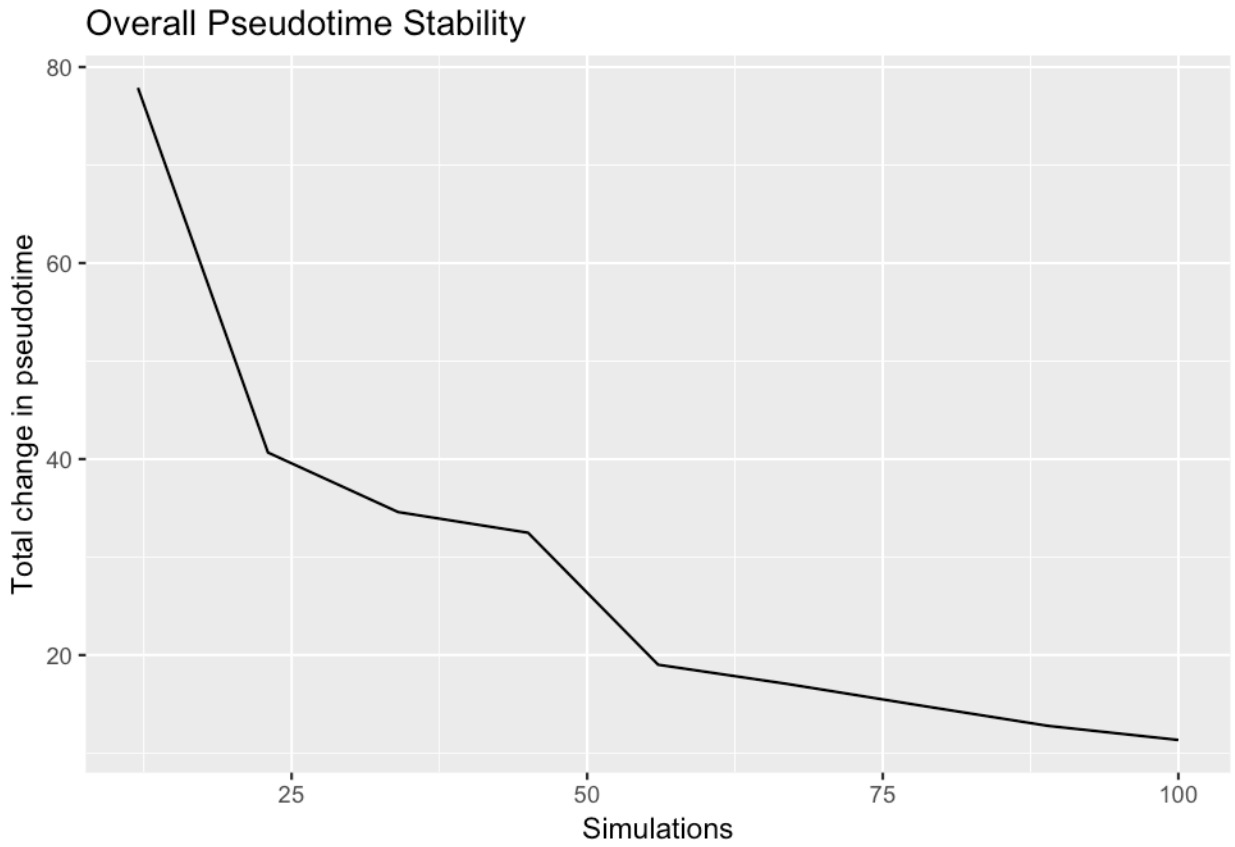
Pseudotime

We perform random simulations to determine the approximate number of transitions to reach each cell in the data from the root and assign it a pseudotime. The random simulations are non-deterministic, so we load our previous result, but the commands run previous were:

```
# Calculate pseudotime floods from the root and load them into the object
flood.result <- floodPseudotime(object, root.cells = root.cells, n = 100, minimum.cells.flooded = 2,
    verbose = T)
```

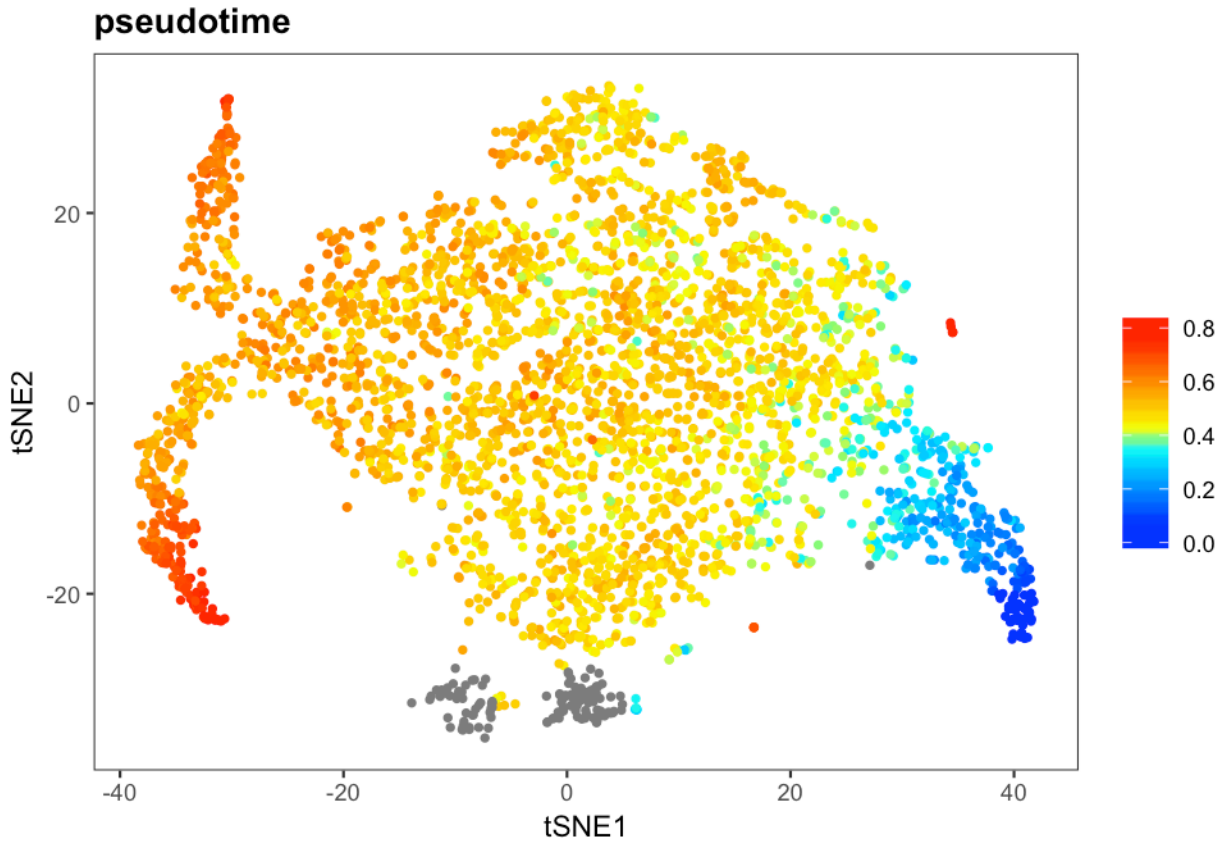
We the process the random simulations to convert them to a 0-1 range pseudotime and verify that enough simulations have been performed.

```
# Process the simulations to determine pseudotime
hydra.en <- floodPseudotimeProcess(hydra.en, flood.result, floods.name = "pseudotime")

# Check that enough pseudotime simulations were run -- is change in pseudotime
# reaching an asymptote?
pseudotimePlotStabilityOverall(hydra.en)
```



Pseudotime was calculated from the foot to the tentacles and hypostome. It has good temporal ordering in all three of these populations, as well as across the body column.

```
# Inspect pseudotime on the tSNE plot
plotDim(hydra.en, "pseudotime")
```



Biased random walks

We tried biased random walks from a couple of tips each from hypostome and tentacle to test that they were well-connected and walks from them actually visited most of the data. The random walk parameters were set rather permissible (optimal 0 cells forward, maximum 500 cells back) because there are many cells in this data set and there are very few cell types. The random simulations are non-deterministic, so we load our previous result, but the commands run previous were:

```
# Bias the transition probabilities by cellular pseudotime
pseudotime.logistic <- pseudotimeDetermineLogistic(hydra.en, pseudotime = "pseudotime",
    optimal.cells.forward = 0, max.cells.back = 500, pseudotime.direction = "<")
tm.biased <- as.matrix(pseudotimeWeightTransitionMatrix(hydra.en, pseudotime = "pseudotime",
    logistic.params = pseudotime.logistic, pseudotime.direction = "<"))

# Simulate biased random walks and load them into the object
walks.en <- simulateRandomWalksFromTips(hydra.en, "tip.clusters", root.cells = root.cells,
    transition.matrix = tm.biased, n.per.tip = walks.to.do, root.visits = 1)

hydra.en <- processRandomWalksFromTips(hydra.en, walks.list = walks.en, verbose = T)
```

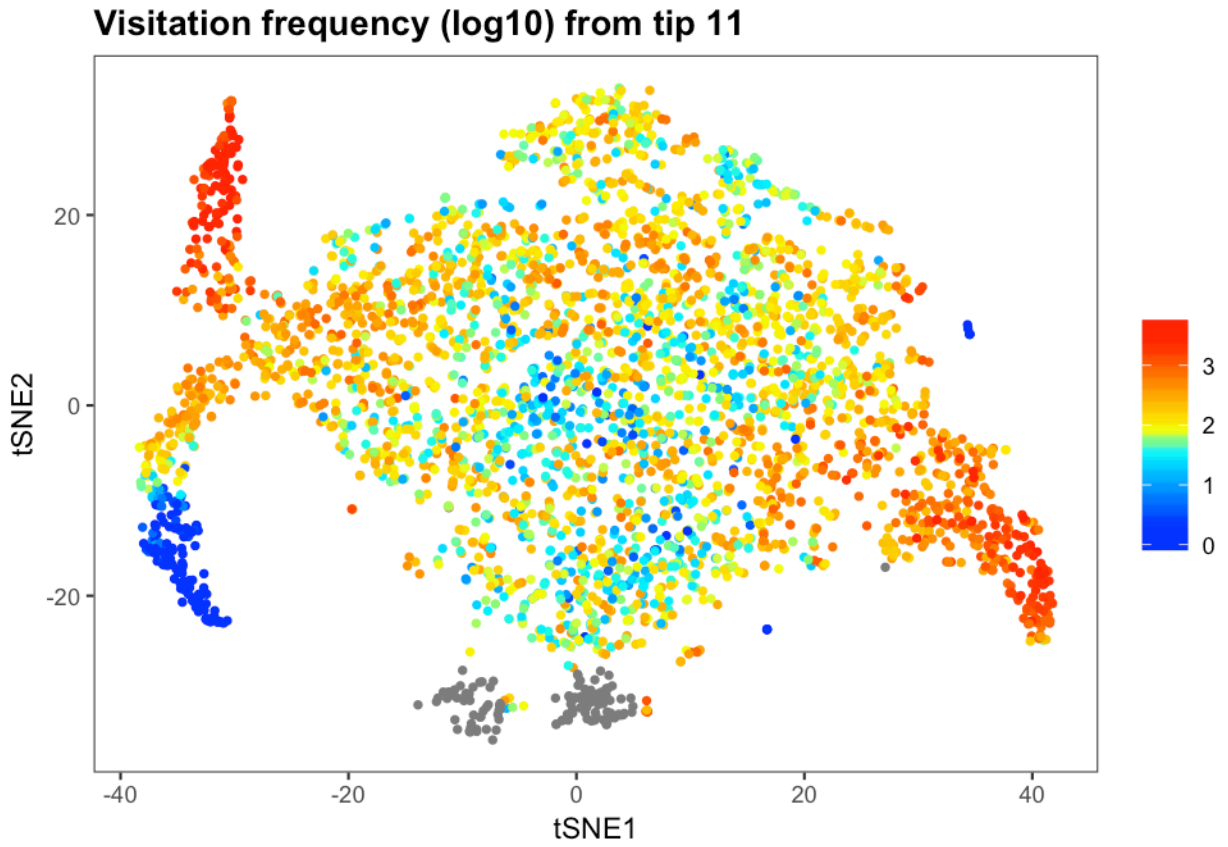
```
## [1] "2018-11-01 16:31:03 - Processing walks from tip 11"
## [1] "Calculating pseudotime with 5000 walks."
## [1] "Calculating pseudotime with 10000 walks."
## [1] "Calculating pseudotime with 15000 walks."
## [1] "Calculating pseudotime with 20000 walks."
## [1] "Calculating pseudotime with 25000 walks."
## [1] "Calculating pseudotime with 30000 walks."
## [1] "Calculating pseudotime with 35000 walks."
## [1] "Calculating pseudotime with 40000 walks."
## [1] "Calculating pseudotime with 45000 walks."
## [1] "Calculating pseudotime with 50000 walks."
## [1] "2018-11-01 16:31:18 - Processing walks from tip 22"
## [1] "Calculating pseudotime with 5000 walks."
## [1] "Calculating pseudotime with 10000 walks."
```

```
## [1] "Calculating pseudotime with 15000 walks."
## [1] "Calculating pseudotime with 20000 walks."
## [1] "Calculating pseudotime with 25000 walks."
## [1] "Calculating pseudotime with 30000 walks."
## [1] "Calculating pseudotime with 35000 walks."
## [1] "Calculating pseudotime with 40000 walks."
## [1] "Calculating pseudotime with 45000 walks."
## [1] "Calculating pseudotime with 50000 walks."
## [1] "2018-11-01 16:31:33 - Processing walks from tip 24"
## [1] "Calculating pseudotime with 5000 walks."
## [1] "Calculating pseudotime with 10000 walks."
## [1] "Calculating pseudotime with 15000 walks."
## [1] "Calculating pseudotime with 20000 walks."
## [1] "Calculating pseudotime with 25000 walks."
## [1] "Calculating pseudotime with 30000 walks."
## [1] "Calculating pseudotime with 35000 walks."
## [1] "Calculating pseudotime with 40000 walks."
## [1] "Calculating pseudotime with 45000 walks."
## [1] "Calculating pseudotime with 50000 walks."
## [1] "2018-11-01 16:31:48 - Processing walks from tip 32"
## [1] "Calculating pseudotime with 5000 walks."
## [1] "Calculating pseudotime with 10000 walks."
## [1] "Calculating pseudotime with 15000 walks."
## [1] "Calculating pseudotime with 20000 walks."
## [1] "Calculating pseudotime with 25000 walks."
## [1] "Calculating pseudotime with 30000 walks."
## [1] "Calculating pseudotime with 35000 walks."
## [1] "Calculating pseudotime with 40000 walks."
## [1] "Calculating pseudotime with 45000 walks."
## [1] "Calculating pseudotime with 50000 walks."
## [1] "2018-11-01 16:32:02 - Processing walks from tip 46"
## [1] "Calculating pseudotime with 5000 walks."
## [1] "Calculating pseudotime with 10000 walks."
## [1] "Calculating pseudotime with 15000 walks."
## [1] "Calculating pseudotime with 20000 walks."
## [1] "Calculating pseudotime with 25000 walks."
## [1] "Calculating pseudotime with 30000 walks."
## [1] "Calculating pseudotime with 35000 walks."
## [1] "Calculating pseudotime with 40000 walks."
## [1] "Calculating pseudotime with 45000 walks."
## [1] "Calculating pseudotime with 50000 walks."
```

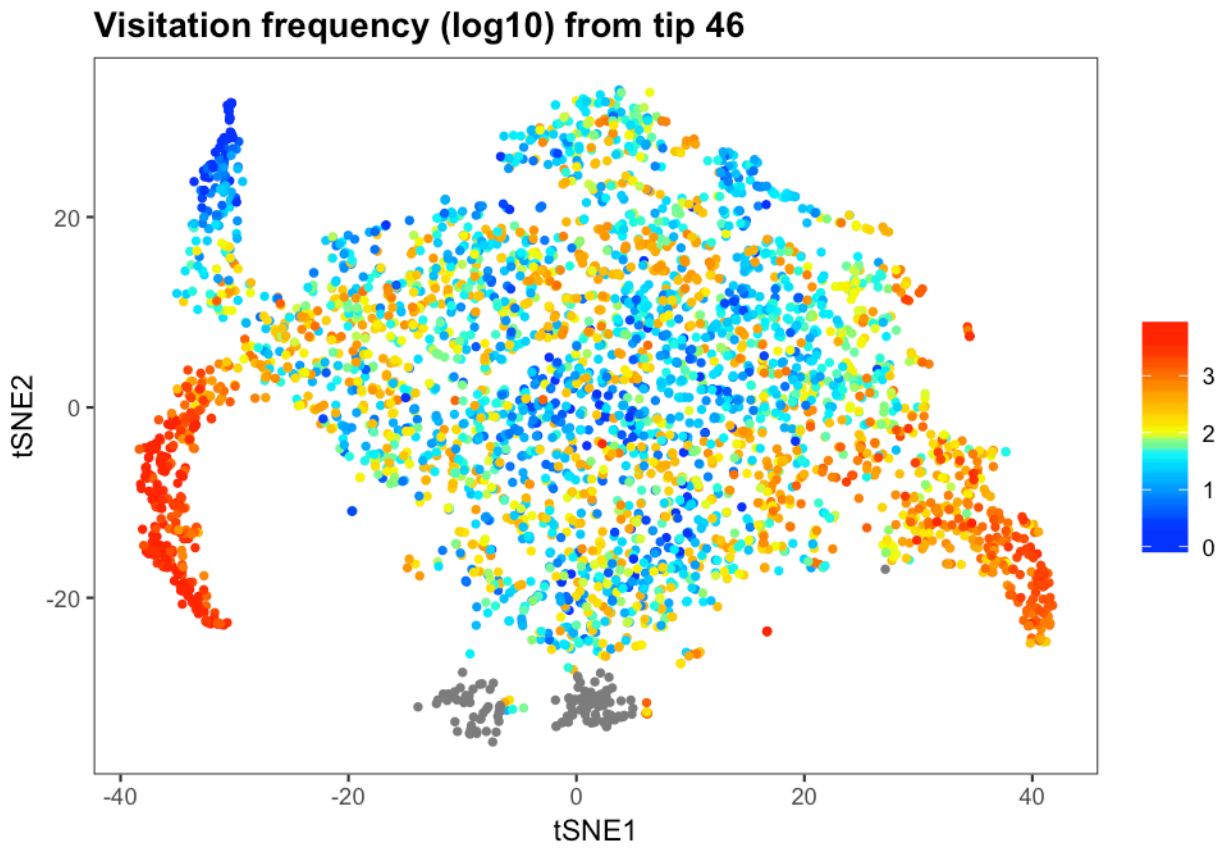
Evaluation of random walks

The random walks from the two tip-most clusters did a good job of visiting the majority of the data, so can just use those for building the tree (11 and 46).

```
# Plot visitation from individual tips
plotDim(hydra.en, "visitfreq.log.11", plot.title = "Visitation frequency (log10) from tip 11")
```




```
plotDim(hydra.en, "visitfreq.log.46", plot.title = "Visitation frequency (log10) from tip 46")
```



Build the tree

Since we have walks with good visitation, we can just build the tree using 11 and 46 as tips.

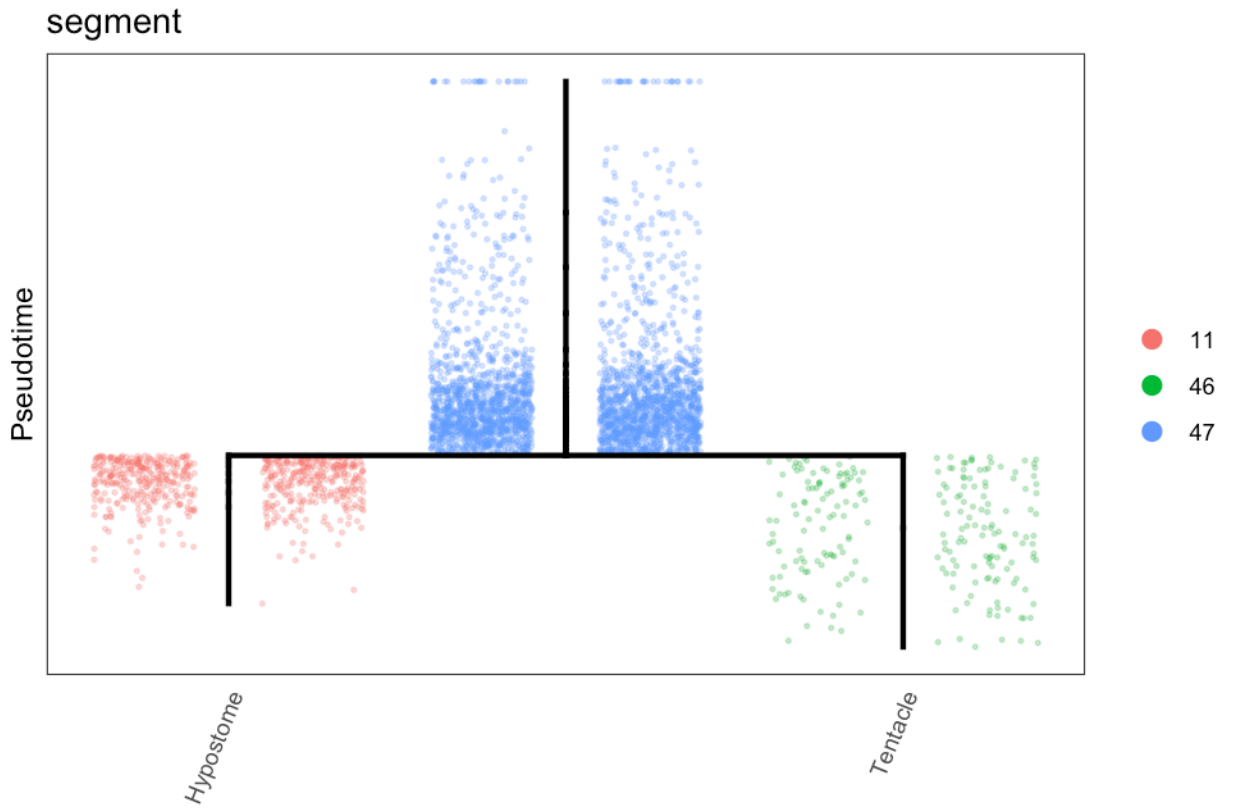
```
# Load the tips into the tree
hydra.en.tree <- loadTipCells(hydra.en, "tip.clusters")

# Select the tips to use for building the tree (since 11 and 46 had good
# visitation, just use those.)
tips.to.build <- c("11", "46")

# Build the tree
hydra.en.tree <- buildTree(hydra.en.tree, pseudotime = "pseudotime", divergence.method = "preference",
    save.all.breakpoint.info = T, tips.use = tips.to.build, cells.per.pseudotime.bin = 25,
    bins.per.pseudotime.window = 8, p.thresh = 0.001, min.cells.per.segment = 10,
    save.breakpoint.plots = NULL)
```

```
## [1] "Calculating divergence between 11 and 46 (Pseudotime 0 to 0.754)"
## [1] "Joining segments 11 and 46 at pseudotime 0.54 to create segment 47"
## [1] "Assigning cells to segments."
## Warning in assignCellsToSegments(object, pseudotime, verbose): 1 cells
## were not visited by a branch that exists at their pseudotime and were not
## assigned.
## [1] "Collapsing short segments."
## [1] "Removing singleton segments."
## [1] "Reassigning cells to segments."
## Warning in assignCellsToSegments(object, pseudotime, verbose): 1 cells
## were not visited by a branch that exists at their pseudotime and were not
## assigned.
## [1] "Assigning cells to nodes."
## [1] "Laying out tree."
## [1] "Adding cells to tree."
```

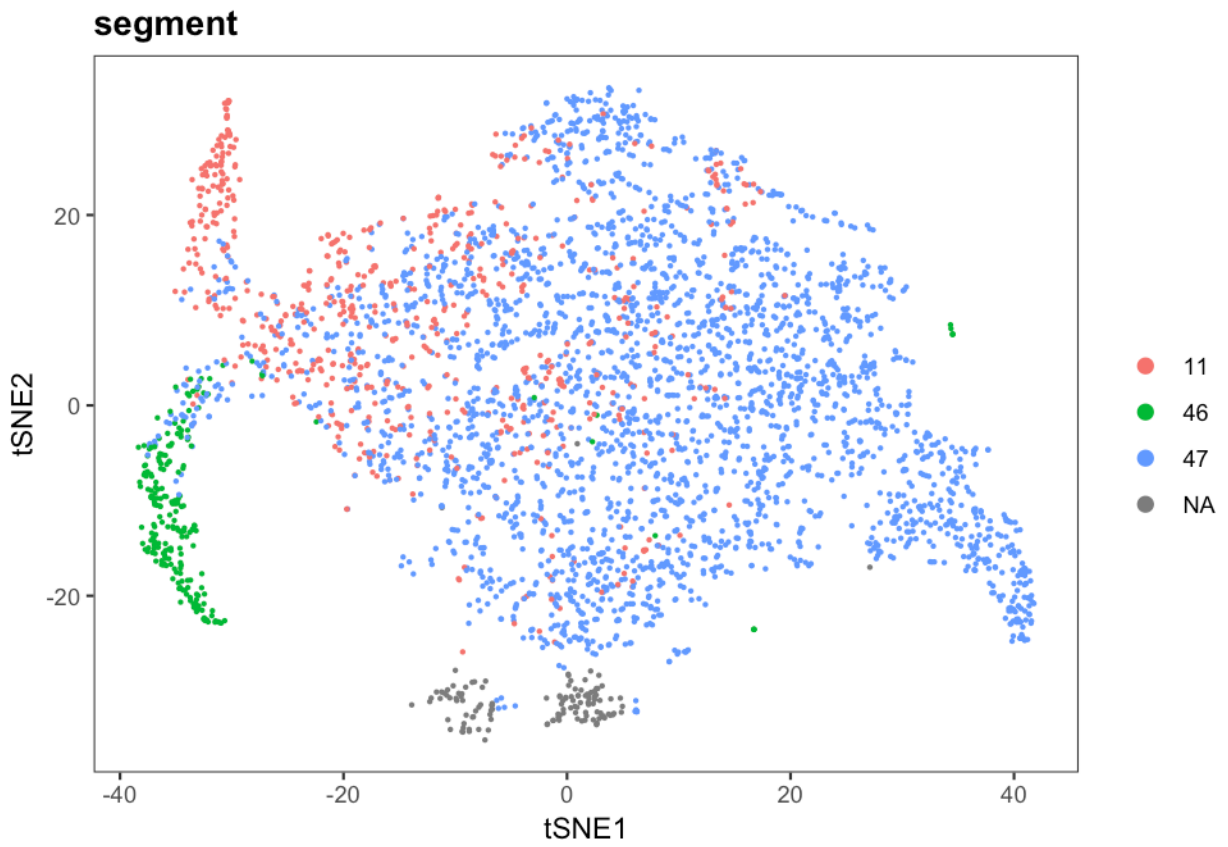
```
# Name the segments to reflect their tissue (identified by external knowledge of
# gene expression)
hydra.en.tree <- nameSegments(object = hydra.en.tree, segments = c("11", "46"), segment.names = c("Hypostome",
    "Tentacle"), short.names = c("Hyp", "Tent"))
```

Visualize the tree

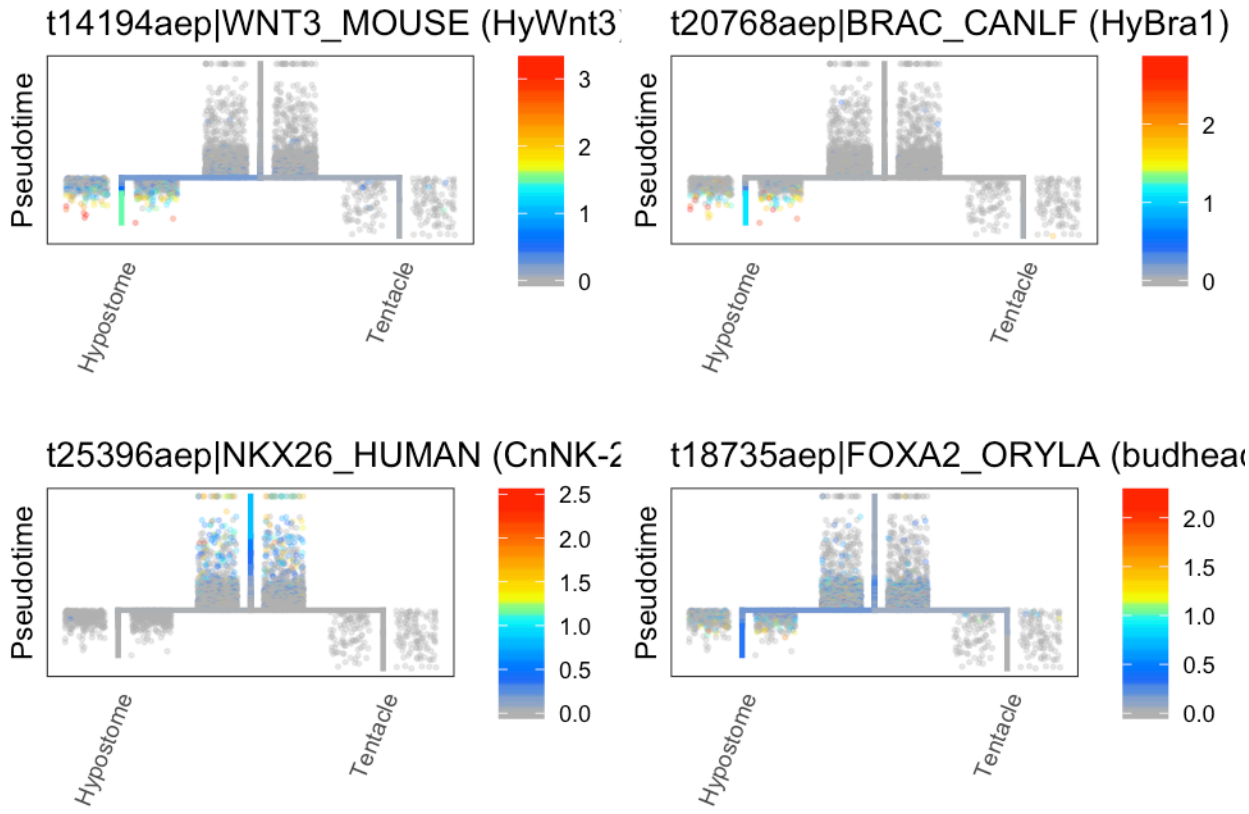
```
plotTree(hydra.en.tree, "segment")
```




```
plotDim(hydra.en.tree, "segment")
```




```
gridExtra::grid.arrange(grobs = list(plotTree(hydra.en.tree, "t14194aep|WNT3_MOUSE",
    title = "t14194aep|WNT3_MOUSE (HyWnt3)"), plotTree(hydra.en.tree, "t20768aep|BRAC_CANLF",
    title = "t20768aep|BRAC_CANLF (HyBra1)"), plotTree(hydra.en.tree, "t25396aep|NKX26_HUMAN",
    title = "t25396aep|NKX26_HUMAN (CnNK-2)"), plotTree(hydra.en.tree, "t18735aep|FOXA2_ORYLA",
    title = "t18735aep|FOXA2_ORYLA (budhead)")))
```



Finding spatially varying genes

Since in this trajectory, "pseudotime" is really a proxy for spatial location (foot to head), we can find the spatially varying genes by finding those that vary in pseudotime. We group cells 5 at a time and calculate a spline curve that fits the mean expression (vs. pseudotime) of each group of 5 cells. We then consider those genes spatially varying that are: (1) expressed (present in at least 1% of cells, mean expression > 0.5 in at least one group of 5 cells, spline curve > 0.5 at some point), (2) vary significantly (their spline curve varies at least 33%), (3) are well fit (noise is usually poorly fit, here we threshold on the sum of squared residuals).

Calculate varying genes

```
# Consider all genes expressed in 1% of endodermal cells
frac.exp <- rowSums(as.matrix(hydra.en.tree@logupx.data > 0))/ncol(hydra.en.tree@logupx.data)
endo.genes <- names(frac.exp)[which(frac.exp > 0.01)]

# Calculate smoothed spline fits of their expression
tentacle.spline <- geneSmoothFit(hydra.en.tree, method = "spline", pseudotime = "pseudotime",
    cells = cellsInCluster(hydra.en.tree, "segment", c("47", "46")), genes = endo.genes,
    moving.window = 1, cells.per.window = 5, spar = 0.9)
```

```
## [1] "2018-11-01 16:32:26: Calculating moving window expression."
## [1] "2018-11-01 16:34:31: Generating un-scaled fits."
## [1] "2018-11-01 16:35:36: Generating scaled fits."
## [1] "2018-11-01 16:36:41: Reducing mean expression data to same dimensions as spline fits."
```

```
hypo.spline <- geneSmoothFit(hydra.en.tree, method = "spline", pseudotime = "pseudotime",
    cells = cellsInCluster(hydra.en.tree, "segment", c("47", "11")), genes = endo.genes,
    moving.window = 1, cells.per.window = 5, spar = 0.9)
```

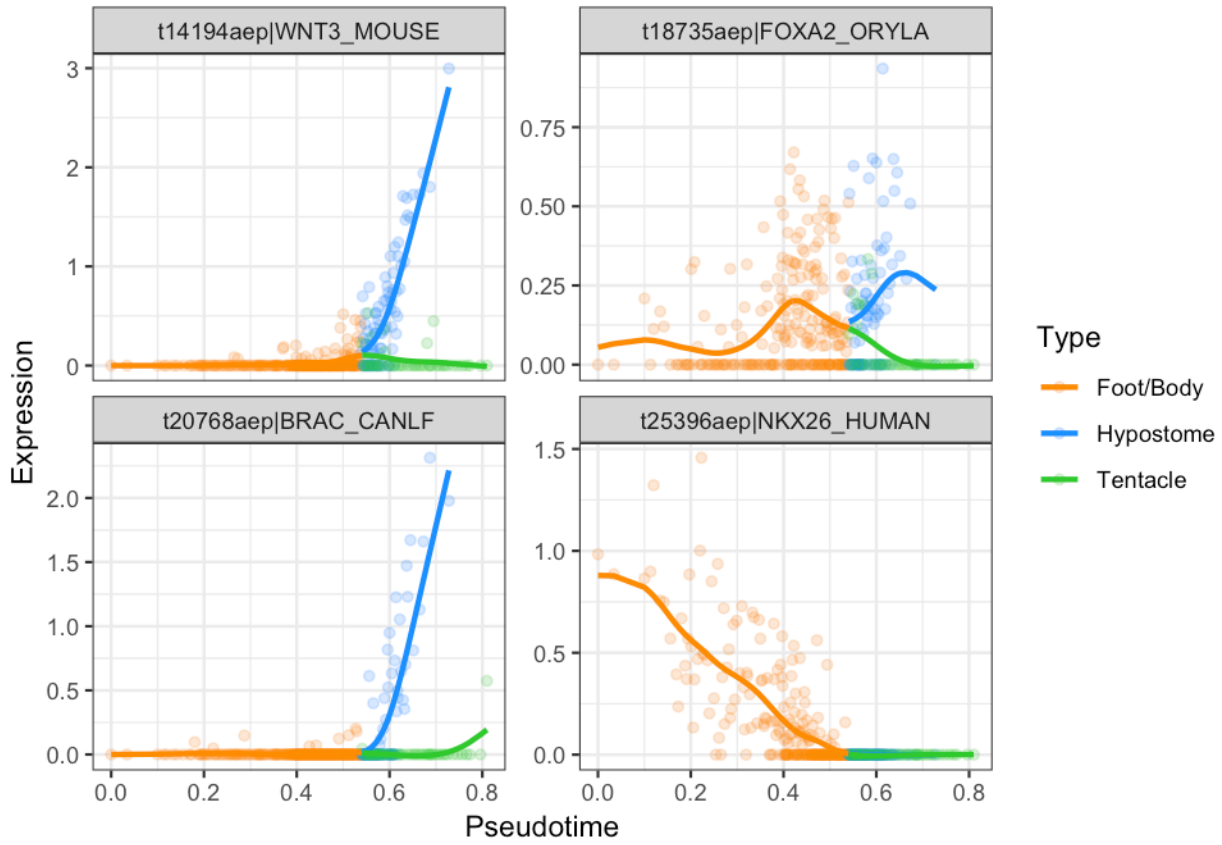
```
## [1] "2018-11-01 16:37:17: Calculating moving window expression."
## [1] "2018-11-01 16:39:42: Generating un-scaled fits."
```

```
## [1] "2018-11-01 16:40:54: Generating scaled fits."
## [1] "2018-11-01 16:42:09: Reducing mean expression data to same dimensions as spline fits."
# Which genes have a spline curve and actual mean expression that crosses 0.5?
tentacle.max.mean.spline <- apply(tentacle.spline$mean.smooth, 1, max)
hypo.max.mean.spline <- apply(hypo.spline$mean.smooth, 1, max)
tentacle.genes.wellexpressed <- names(which(tentacle.max.mean.spline > 0.5))
hypo.genes.wellexpressed <- names(which(hypo.max.mean.spline > 0.5))

# Which genes change in their scaled log2 mean expression value by at least 33%?
tentacle.change.scale <- apply(tentacle.spline$scaled.smooth, 1, function(x) diff(range(x)))
hypo.change.scale <- apply(hypo.spline$scaled.smooth, 1, function(x) diff(range(x)))
tentacle.genes.scale <- names(which(tentacle.change.scale >= 0.33))
hypo.genes.scale <- names(which(hypo.change.scale >= 0.33))

# Which genes are well fit by the spline curves? (Noise is usually poorly fit by
# a curve)
tentacle.spline.fit <- apply(tentacle.spline$scaled.smooth - tentacle.spline$scaled.expression.red,
    1, function(i) sum(i^2))
hypo.spline.fit <- apply(hypo.spline$scaled.smooth - hypo.spline$scaled.expression.red,
    1, function(i) sum(i^2))
tentacle.spline.fit.norm <- tentacle.spline.fit/ncol(tentacle.spline$scaled.smooth)
hypo.spline.fit.norm <- hypo.spline.fit/ncol(hypo.spline$scaled.smooth)
tentacle.wellfit <- names(which(tentacle.spline.fit.norm <= 0.02))
hypo.wellfit <- names(which(hypo.spline.fit.norm <= 0.02))

# Take the intersection of those genes and use them as the 'varying genes' in the
# heatmap & analysis
tentacle.genes <- intersect(intersect(tentacle.genes.scale, tentacle.wellfit), tentacle.genes.wellexpressed)
hypo.genes <- intersect(intersect(hypo.genes.scale, hypo.wellfit), hypo.genes.wellexpressed)

## Combine pieces of spline fits into a single list for multi-plotting Identify
## the pseudotime of the branchpoint
pt.crop <- as.numeric(unlist(hydra.en.tree@tree$segment.pseudotime.limits)[1])
# Crop according to the pseudotime of the branchpoint
foot.only.spline <- cropSmoothFit(tentacle.spline, pt.max = pt.crop)
hypostome.only.spline <- cropSmoothFit(hypo.spline, pt.min = pt.crop)
tentacle.only.spline <- cropSmoothFit(tentacle.spline, pt.min = pt.crop)
# Combine into a list; Names in the plots are determined by names of the smooth
# objects in the list
splines <- list(foot.only.spline, hypostome.only.spline, tentacle.only.spline)
names(splines) <- c("Foot/Body", "Hypostome", "Tentacle")
```

Spline plots

We can then plot the expression of genes and their fit splines.

```
endoderm.genes.plot <- c("t14194aep|WNT3_MOUSE", "t20768aep|BRAC_CANLF", "t25396aep|NKX26_HUMAN",
    "t18735aep|FOXA2_ORYLA")

plotSmoothFitMultiCascade(smoothed.fits = splines, genes = endoderm.genes.plot, scaled = F,
    colors = c(`Foot/Body` = "#FF8C00", Hypostome = "#1E90FF", Tentacle = "#32CD32"))
```



Heatmaps

Finally, we can make heatmaps of the expression of all of the genes that we found to vary spatially. These we save as PDF output because they do not perform well inside of R Markdown.

```
# Heatmap Basics
cols <- (scales::gradient_n_pal(RColorBrewer::brewer.pal(9, "YlOrRd")))(seq(0, 1,
    length.out = 50))

# Get the scaled data, z-score it, and set values <0 to 0, the hierarchical
# cluster.  This emphasizes clustering on regions of peak expression for a gene.
se <- tentacle.spline$scaled.expression[tentacle.genes, ]
se.sd <- apply(se, 1, sd)
se.mean <- apply(se, 1, mean)
se.z <- sweep(sweep(se, 1, se.mean, "-"), 1, se.sd, "/")
se.z[se.z < 0] <- 0
h.sez <- as.dendrogram(hclust(dist(se.z)))

# Find a 'peak pseudotime' for each gene by taking the weighted average of each
# window's pseudotime, weighted by the expression z-score (>0) within it Can use
# this to reorder the dendrogram to try to put it in pseudotime order.
pt.wm <- apply(se.z, 1, function(w) {
    weighted.mean(x = as.numeric(colnames(se.z)), w = w)
})

# Generate the actual heatmap and save as a PDF.
font.size <- 0.08
pdf(paste0(main.path, "URD/Endoderm/Endoderm-Varying-Tentacle.pdf"), width = 17,
    height = 22)
gplots::heatmap.2(as.matrix(se), Rowv = reorder(h.sez, max(pt.wm) - pt.wm, agglo.FUN = median),
    Colv = F, dendrogram = "none", col = cols, trace = "none", density.info = "none",
    key = F, cexCol = 0.8, cexRow = font.size, margins = c(5, 8), lwid = c(0.3, 4),
    lhei = c(0.4, 4), labCol = NA)
title("Endoderm: Foot to Tentacle", line = -1, adj = 0.48, cex.main = 4)
```

```
dev.off()
```

```
## quartz_off_screen
##                 2
```

```
# Heatmap Basics
cols <- (scales::gradient_n_pal(RColorBrewer::brewer.pal(9, "YlOrRd")))(seq(0, 1,
    length.out = 50))

# Get the scaled data, z-score it, and set values <0 to 0, the hierarchical
# cluster.  This emphasizes clustering on regions of peak expression for a gene.
se <- hypo.spline$scaled.expression[hypo.genes, ]
se.sd <- apply(se, 1, sd)
se.mean <- apply(se, 1, mean)
se.z <- sweep(sweep(se, 1, se.mean, "-"), 1, se.sd, "/")
se.z[se.z < 0] <- 0
h.sez <- as.dendrogram(hclust(dist(se.z)))

# Find a 'peak pseudotime' for each gene by taking the weighted average of each
# window's pseudotime, weighted by the expression z-score (>0) within it Can use
# this to reorder the dendrogram to try to put it in pseudotime order.
pt.wm <- apply(se.z, 1, function(w) {
    weighted.mean(x = as.numeric(colnames(se.z)), w = w)
})

# Generate the actual heatmap and save as a PDF.
font.size <- 0.08
pdf(paste0(main.path, "URD/Endoderm/Endoderm-Varying-Hypostome.pdf"), width = 17,
    height = 22)
gplots::heatmap.2(as.matrix(se), Rowv = reorder(h.sez, max(pt.wm) - pt.wm, agglo.FUN = median),
    Colv = F, dendrogram = "none", col = cols, trace = "none", density.info = "none",
    key = F, cexCol = 0.8, cexRow = font.size, margins = c(5, 8), lwid = c(0.3, 4),
    lhei = c(0.4, 4), labCol = NA)
title("Endoderm: Foot to Hypostome", line = -1, adj = 0.48, cex.main = 4)
dev.off()
```

```
## quartz_off_screen
##                 2
```

Save results

```
write(tentacle.genes, file = paste0(main.path, "URD/Endoderm/Genes-Endoderm-Varying-Tentacle.txt"))
write(hypo.genes, file = paste0(main.path, "URD/Endoderm/Genes-Endoderm-Varying-Hypostome.txt"))
saveRDS(splines, file = paste0(main.path, "URD/Endoderm/Splines-Endoderm.rds"))
saveRDS(hydra.en.tree, file = paste0(main.path, "URD/Endoderm/Hydra_URD_Endoderm.rds"))
```