Bioconductor workflow for single-cell RNA-seq data analysis: dimensionality reduction, clustering, and pseudotime ordering

Author Name1[1] **and Author Name2**[2]

[1]**Address of author1**
[2]**Address of author2**

Abstract Abstracts should be up to 300 words and provide a succinct summary of the article. Although the abstract should explain why the article might be interesting, care should be taken not to inappropriately over-emphasise the importance of the work described in the article. Citations should not be used in the abstract, and the use of abbreviations should be minimized.

Keywords
Please list up to eight keywords to help readers interested in your article find it more easily.

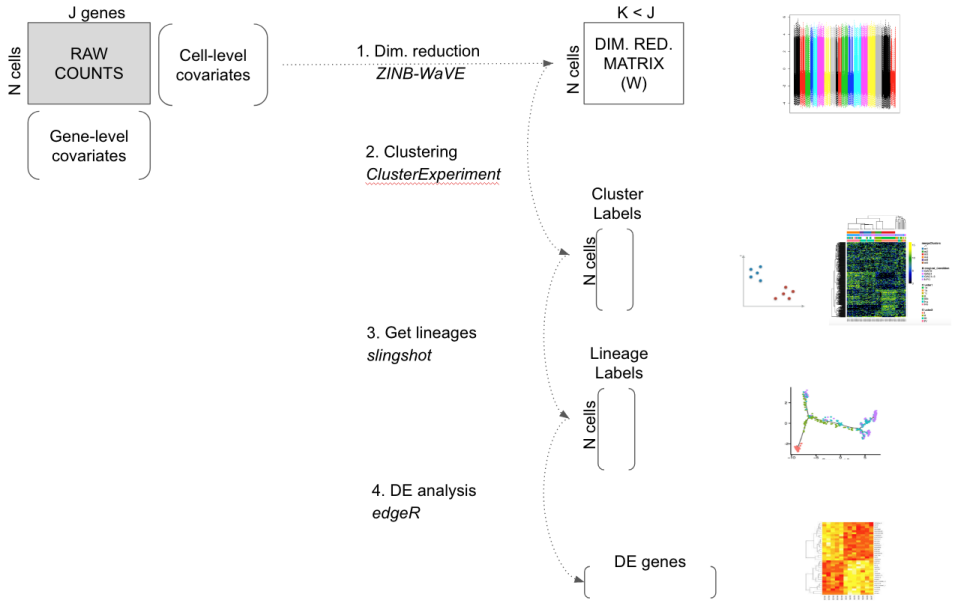


Figure 1. Our workflow

```
runZinb <- TRUE
runClus <- TRUE
NCORES <- 2
mysystem = Sys.info()[['sysname']]
if (mysystem == 'Darwin'){
  registerDoParallel(NCORES)
  register(DoparParam())
}else if (mysystem == 'Linux'){
  register(bpstart(MulticoreParam(workers=NCORES)))
}else{
  print('Please change this to allow parallel computing')
  register(SerialParam())
}
```

EAP: small request. Can everyone put a line between the beginning of a r chunk and text? It makes it nicely formated for my text editor.

Introduction

- growing interest in single-cell RNA-seq data. New statistical tools need to be developped compared to bulk RNA-seq because bias: drop-outs, amplification bias. There is also increasing size of datasets (10X-genomics, 1.3M cells) -> need of integrated workflows of new statistical tools
- previously developped : F1000 Research bioc (A. T. Lun, McCarthy, and Marioni 2016) and scater (McCarthy et al. 2017) are pipelines containing several useful methods for quality control, visualisation and pre-processing of data. In these pipelines, single-cell expression data are organized in objects of the SCESet class which allows integrated workflows. However, they are used mostly to prepare the data for further downstream analysis and do not focus on steps like clustering, pseudotime ordering, and DE analysis.
- we propose an integrated workflow with dowstream analysis: dimensionality reduction adjusting for gene and cell-level covariates, robust and stable clustering using resumpling, pseudotime ordering, and DE analysis for the clusters found. We use a summarizedExperiment object to have an integrated pipeline. We also propose useful functions for visualization.

```
knitr::include_graphics('schema_workflow.png')
```

Analysis of olfactory stem cell lineages using single cell RNA-seq data

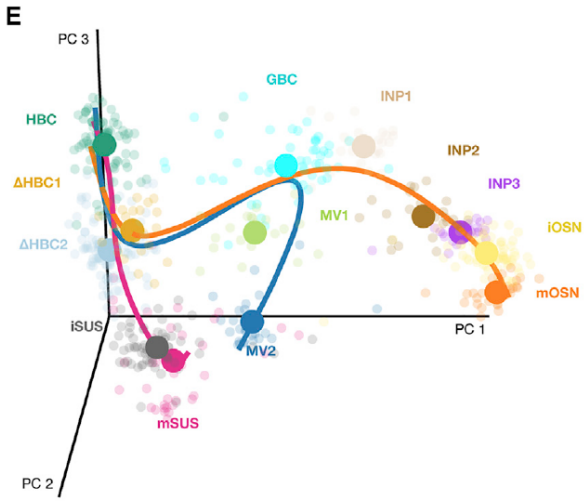


Figure 2. Stem cell differentiation in the mouse olfactory epithelium. Figure 2e from Fletcher et al paper.

Overview

The workflow is illustrated using data from a scRNA-seq study of stem cell differentiation in the mouse olfactory epithelium (Fletcher et al. 2017). The olfactory epithelium contains mature olfactory sensory neurons that are continuously renewed in the epithelium via neurogenesis through differentiation of globose basal cells (GBCs) which are the actively proliferating cells in the epithelium. When a severe injury to the entire tissue happens, the olfactory epithelium can regenerate from normally quiescent stem cells called horizontal basal cells (HBCs) which become activated to differentiate and reconstitute all major cell types in the epithelium.
The dataset we work with in this workflow was generated to study the differentitation from the HBCs stem cells to different cell types present in the epithelium. To map the developmental trajectories of the multiple cell lineages arising from the olfactory epithelium's HBC stem cell, different lineages were assigned to each cell in the dataset and results were confirmed experimentally using in vivo lineage tracing. It was found that the first major bifurcation in the HBC lineage trajectory occurs prior to cell division, producing either sustentacular (support) cells or GBCs. Then, the GBC lineage, in turn, branches to give rise to olfactory sensory neurons, microvillous cells, and cells of the Bowman's gland.

```
knitr::include_graphics('stemcelldiff_Fletcher2017_2e.png')
```

Generation of the data

The dataset we use for the workflow is publicly available at `https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE95601&format=file&file=GSE95601%5FoeHBCdiff%5FCufflinks%5FeSet%2ERda%2Egz`. It contains 849 cells and 28361 detected genes. Each cell has been labelled with labels publicly available at `https://raw.githubusercontent.com/rufletch/p63-HBC-diff/master/ref/oeHBCdiff_clusterLabels.txt`. Pre-processing steps are as follow: - ERCC and CreER cells were removed, - low-quality cells were removed using package scone (ADD REF) and
FP: we should add details here.
Finally to speed up the computations, we kept only the 1000 most variable genes in this analysis. After the preprocessing steps, the dataset we are working with has 1000 genes and 747 cells. Along the workflow, we use a SummarizedExperiement object to keep the counts and the metadata in the same object

```
if(!file.exists("../data/oe_se_1000Var.rda")) {
  source("createData.R")
}

load("../data/oe_se_1000Var.rda")
core
```

```
## class: SummarizedExperiment
## dim: 1000 747
## metadata(0):
## assays(1): counts
## rownames(1000): Cbr2 Cyp2f2 ... Rnf13 Atp7b
```

```
## rowData names(0):
## colnames(747): OEP01_N706_S501 OEP01_N701_S501 ... OEL23_N704_S503
##   OEL23_N703_S502
## colData names(19): Experiment Batch ... CreER ERCC_reads
```

```
batch <- colData(core)$Batch
```

Metadata for the cells is stored in colData from the SummarizedExperiment object. Cells have been processed in 18 different batches

```
col_batch = rep(brewer.pal(9, "Set1"), 2)
names(col_batch) = unique(batch)
table(batch)
```

```
## batch
## GBC08A GBC08B GBC09A GBC09B    P01    P02   P03A   P03B    P04    P05
##     39     40     35     22     31     48     51     40     20     23
##    P06    P10    P11    P12    P13    P14    Y01    Y04
##     51     40     50     50     60     47     58     42
```

We also have access to the quality control measures that were used to filter low quality cells

```
qc <- colData(core)[, !names(colData(core)) %in% c("Batch", "Experiment", "clusterLabels")]
head(qc, 2)
```

```
## DataFrame with 2 rows and 16 columns
##                    NREADS  NALIGNED    RALIGN TOTAL_DUP    PRIMER
##                 <numeric> <numeric> <numeric> <numeric> <numeric>
## OEP01_N706_S501   3313260   3167600   95.6035   47.9943 0.0154566
## OEP01_N701_S501   2902430   2757790   95.0167   45.0150 0.0182066
##                 PCT_RIBOSOMAL_BASES PCT_CODING_BASES PCT_UTR_BASES
##                           <numeric>        <numeric>     <numeric>
## OEP01_N706_S501              2e-06         0.200130      0.230654
## OEP01_N701_S501              0e+00         0.182461      0.201810
##                 PCT_INTRONIC_BASES PCT_INTERGENIC_BASES PCT_MRNA_BASES
##                          <numeric>            <numeric>      <numeric>
## OEP01_N706_S501           0.404205             0.165009       0.430784
## OEP01_N701_S501           0.465702             0.150027       0.384271
##                 MEDIAN_CV_COVERAGE MEDIAN_5PRIME_BIAS MEDIAN_3PRIME_BIAS
##                          <numeric>          <numeric>          <numeric>
## OEP01_N706_S501           0.843857           0.061028           0.521079
## OEP01_N701_S501           0.914370           0.033350           0.373993
##                     CreER ERCC_reads
##                 <numeric>  <numeric>
## OEP01_N706_S501         1      10516
## OEP01_N701_S501      3022       9331
```

```
clus.labels <- colData(core)[, "clusterLabels"]
```

In original work (Fletcher et al. 2017), cells have been clustered into 14 different clusters

```
col_clus <- c("transparent", brewer.pal(12, "Set3"), brewer.pal(8, "Set2"))
col_clus <- col_clus[1:length(unique(clus.labels))]
names(col_clus) <- sort(unique(clus.labels))
table(clus.labels)
```

```
## clus.labels
##  -2   1   2   3   4   5   7   8   9  10  11  12  14  15
## 151  90  25  54  35  93  58  27  74  26  21  35  26  32
```

Note that in this dataset batches are somehow confounded with the clusters found in the original work.

```
table(data.frame(batch = as.vector(batch),
                 cluster = clus.labels))
```

```
##         cluster
## batch    -2  1  2  3  4  5  7  8  9 10 11 12 14 15
##   GBC08A  3  0  2 12  9  0  0  0  0  0  2  0  2  9
##   GBC08B  8  0  7  5  3  0  0  0  1  2  3  0  5  6
##   GBC09A  6  0  1  5  8  0  0  0  1  1  0  0  6  7
##   GBC09B 12  0  2  1  3  0  0  0  1  0  0  0  3  0
##   P01     7  0  2  4  3 15  0  0  0  0  0  0  0  0
##   P02     5  2  0  9  3 15  3  3  2  3  0  2  1  0
##   P03A   15  3  0  2  0 12  2  9  4  2  0  2  0  0
##   P03B    9  1  2  1  1 11  1  2  8  1  1  2  0  0
##   P04     8  0  0  0  0  9  1  0  1  1  0  0  0  0
##   P05     3  0  0  0  1 11  3  0  1  0  2  2  0  0
##   P06    12  1  2  3  0  8  2  4  8  4  1  2  2  2
##   P10     7  3  1  4  0  3  5  8  1  0  2  5  0  1
##   P11     6  2  1  1  0  1  5  1 22  3  1  6  0  1
##   P12    10  0  2  0  0  4 10  0  8  2  3  6  4  1
##   P13    13  1  2  4  0  4 15  0  4  5  6  1  3  2
##   P14     9  0  0  1  2  0 11  0 12  2  0  7  0  3
##   Y01     8 46  1  1  2  0  0  0  0  0  0  0  0  0
##   Y04    10 31  0  1  0  0  0  0  0  0  0  0  0  0
```

Dimensionality reduction adjusting for batches

In single-cell RNA-seq analysis, dimensionality reduction is often used as a preliminary step prior to downstream analysis where clustering, pseudotime ordering, or differential expression analysis are performed. It allows the data to become more tractable, both from a statistical (cf. curse of dimensionality) and computational point of view. Additionally, technical noise can be reduced while preserving the often intrinsically low dimensional signal of interest.

Here, we perform dimensionality reduction using bioconductor package zinbwave which allows to fit a zero-inflated negative binomial model (ZINB-WaVE) to get a low-dimensional representation of the data while accounting for zero inflation (dropouts), over-dispersion, and the count nature of the data. The model can include a sample-level intercept, which serves as a global-scaling normalization factor. The user can also include both gene-level and sample-level covariates. The inclusion of observed and unobserved sample-level covariates enables normalization for complex, non-linear effects (often referred to as batch effects), while gene-level covariates may be used to adjust for sequence composition effects, such as gene length and GC-content effects.

As with most of the dimensionality reduction methods, the user needs to specify the number of dimensions of the new low dimensional space. Here, we use 50 dimensions and we adjust for the batches.

FP: do we want to expose the user to the equations? In the one hand, it would be easier to explain what we are looking at (W, normalized values, ...). In the other hand, it would need a good amount of explanations. I have a preference to show the equations, but what do you think?

```
fn <- '../data/zinb_batch.rda'
if (runZinb & !file.exists(fn)){
  print(system.time(se <- zinbDimRed(core, K = 50, X = '~ Batch',
                                     residuals = TRUE,
                                     normalizedValues = TRUE)))
  save(se, file = fn)
}else{
  load(fn)
}
```

DR: the chunk above and the similar one for clusterExperiment are fine for now, but in the published workflow, we should just rely on the markdown cache system (the code above doesn't check for changes to the file or code – just that a version of the file exists).

Normalized values

When function zinbDimRed is called, normalized values of the counts can be computed and used for visualization (e.g. in heatmaps). The normalized values are the residuals of the fit of our ZINB model where the gene and cell covariates were the same as the user specified in the zinbDimRed call but the number of unknown covariates is set to be null (i.e. K=0). When no unknown covariates are included in the model, the residuals represents the counts adjusted for the gene and cell covariates. To compute the residuals, we use the deviance residuals.

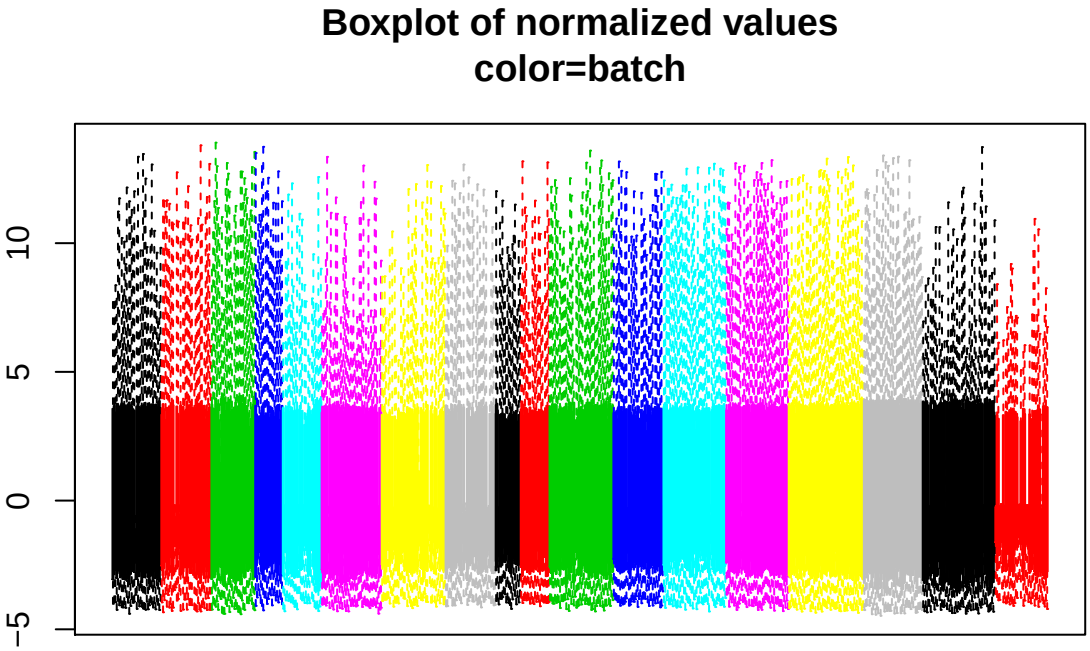
FP: note to myself: why do we have infinite values in the residuals now? It shows the same results as before, but we should not see infinite values here!

```
norm <- assays(se)$normalizedValues
if (sum(is.infinite(norm))>0){
  maxNorm = max(norm[!is.infinite(norm)])
  assays(se)$normalizedValues[is.infinite(norm)] <- maxNorm
  norm <- assays(se)$normalizedValues
}
norm[1:3,1:3]
```

```
##        OEP01_N706_S501 OEP01_N701_S501 OEP01_N707_S507
## Cbr2          4.557371        4.375069       -4.142697
## Cyp2f2        4.321644        4.283266        4.090283
## Gstm1         4.796498        4.663366        4.416324
```

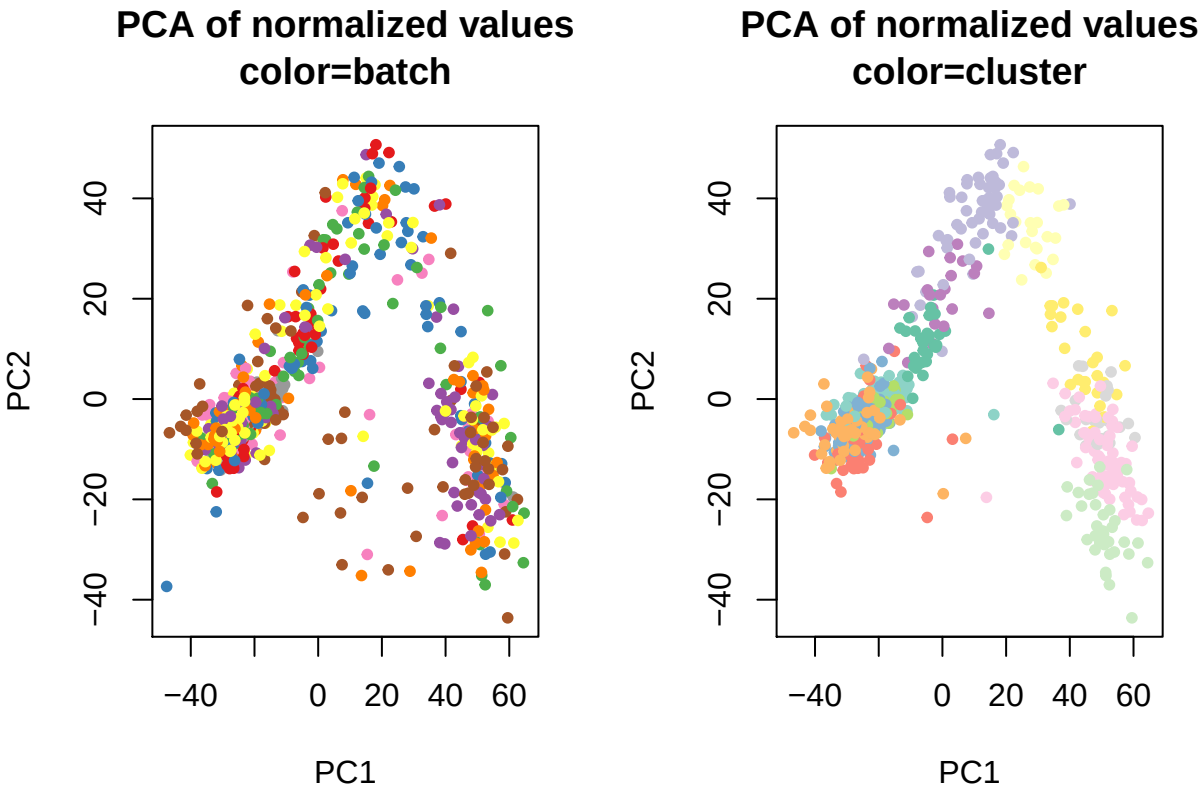
As expected, the distributions of the normalized counts are independent of the batches

```
norm_order <- norm[, order(as.numeric(batch))]
col_order <- as.numeric(batch)[order(as.numeric(batch))]
boxplot(norm_order, main='Boxplot of normalized values\ncolor=batch',
        col = col_order, staplewex = 0, outline = 0, border = col_order,
        xaxt = 'n')
```



We performed principal component analysis (PCA) on the normalized values where colors are for batches on the left and original clusters on the right. As expected, there are no clustering on the left side and clustering on the right side.

```
pca <- prcomp(t(norm))
par(mfrow = c(1,2))
plot(pca$x, col = col_batch[batch], pch = 20,
     main="PCA of normalized values\ncolor=batch")
plot(pca$x, col = col_clus[as.character(clus.labels)], pch = 20,
     main = "PCA of normalized values\ncolor=cluster")
```



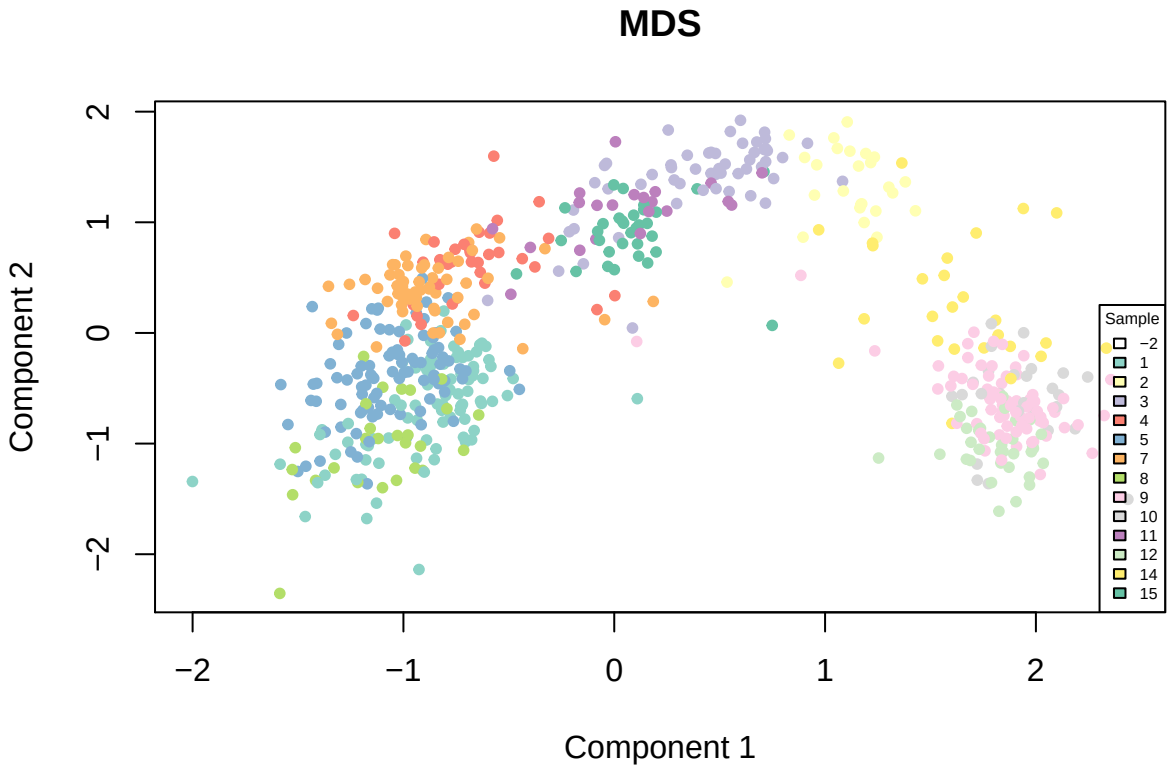

```
par(mfrow = c(1,1))
```

Overall, looking at the normalized values, it seems that the batch effect was removed.

Dimensionality reduction

Calling function zinbDimRed, we performed dimensionality reduction with K = 50. We can visualize the low dimensional matrix performing multidimensional scaling (MDS) in two dimensions. To make sure that the low dimensional matrix captured the signal of interest, the colors shown here are from the original clusters.

```
W <- colData(se)[, grepl('^W', colnames(colData(se)))]
W <- as.matrix(W)
d <- dist(W)
fit <- cmdscale(d, eig = TRUE, k = 2)
plot(fit$points, col = col_clus[as.character(clus.labels)], main = 'MDS', pch = 20,
     xlab = 'Component 1', ylab = 'Component 2')
legend(x = 'bottomright', legend = unique(names(col_clus)), cex = .5,
       fill = unique(col_clus), title = 'Sample')
```



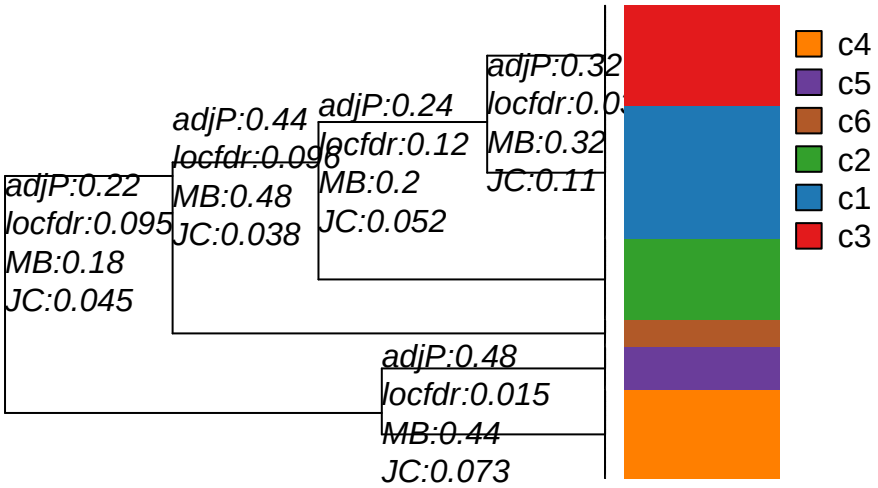
Clustering of the cells

We use clusterExperiment with W.
EP: I updated it to work on a SE object so that it has the meta data. If you have a summarized experiment object with W already, you could use that as long as assay(seObj) gives W.

```
fn <- '../data/RSEC_W_combineMinSize10.rda'
if (runClus & !file.exists(fn)){
  #symbol for samples missing from original clustering
  seObj <- SummarizedExperiment(t(W), colData = colData(core))
  print(system.time(ceObj <- RSEC(seObj, k0s = 4:15, alphas = c(0.1),
                                  betas = 0.8, dimReduce="none",
                clusterFunction = "hierarchical01", minSizes=1,
                ncores = NCORES, isCount=FALSE,
                subsampleArgs = list(resamp.num=100,
                                     clusterFunction="kmeans",
                                     clusterArgs=list(nstart=10)),
                seqArgs = list(k.min=3, top.can=5), verbose=TRUE,
                combineProportion = 0.7,
                mergeMethod = "none", random.seed=424242,
                combineMinSize = 10)))
  save(ceObj, file = fn)
}else{
  load(fn)
}
```

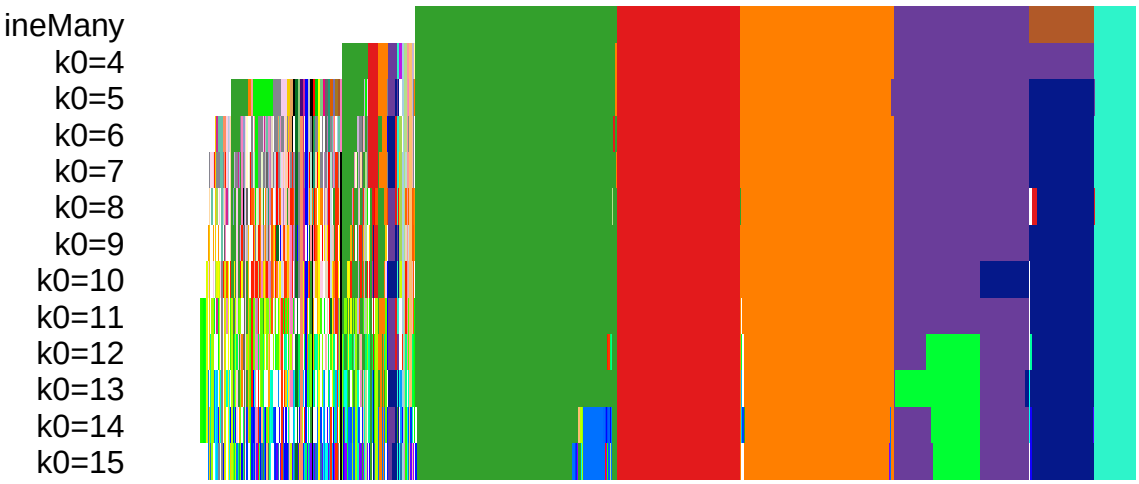
```
## Warning in .makeColors(clusters, colors = bigPalette): too many clusters to
## have unique color assignments

## Warning in .makeColors(clusters, colors = bigPalette): too many clusters to
## have unique color assignments

## Note: Merging will be done on ' combineMany ', with clustering index 1

## Warning: glm.fit: fitted rates numerically 0 occurred

## Warning: glm.fit: fitted rates numerically 0 occurred

## Warning: glm.fit: fitted rates numerically 0 occurred
```

```
## Warning in .makeColors(clusters, colors = bigPalette): too many clusters to
## have unique color assignments
```



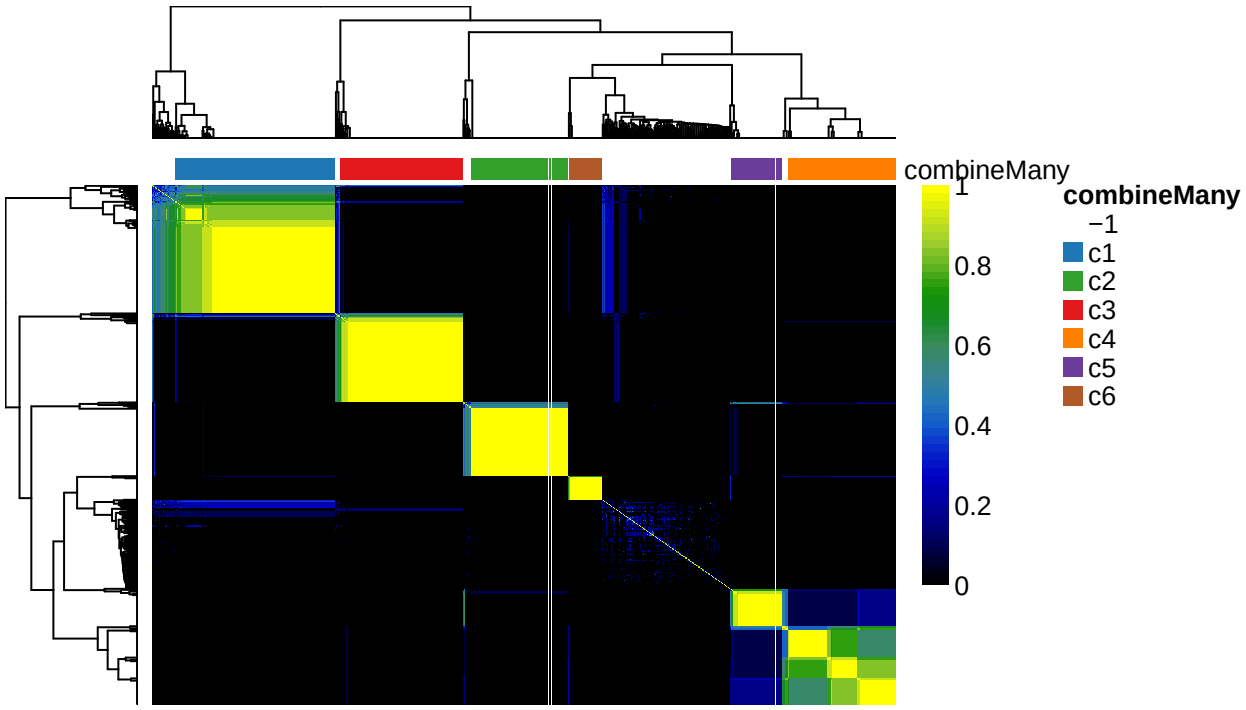

```
##     user   system  elapsed
## 4650.537  195.296 5300.856
```

```
plotClusters(ceObj, colPalette = c(bigPalette, rainbow(199)))
```




```
plotCoClustering(ceObj)
```

```
## Warning in .makeColors(clusters, colors = bigPalette): too many clusters to
## have unique color assignments
```




```
table(primaryClusterNamed(ceObj))
```

```
##
##  -1  c1  c2  c3  c4  c5  c6
## 172 161  98 123 108  52  33
```

```
sum(primaryCluster(ceObj) == -1)
```

```
## [1] 172
```

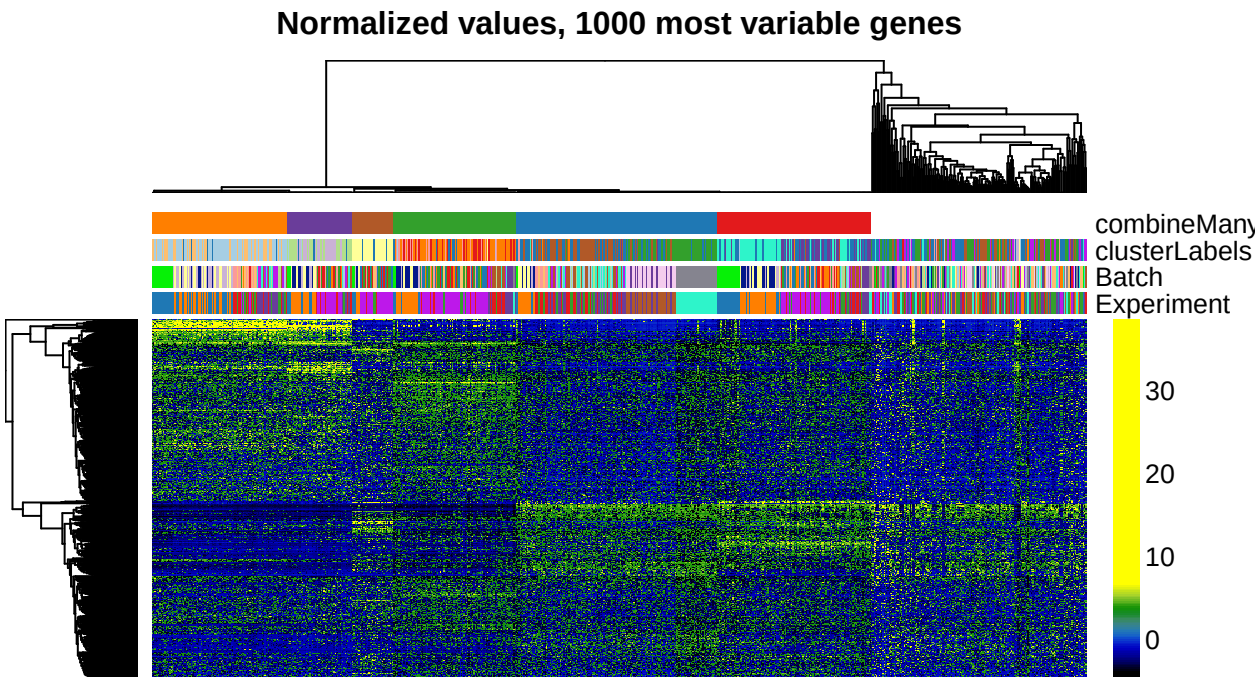
FP: Elizabeth, we are working with the W here, does the locfdr make sense in this context? I set eval=FALSE in the next chunk to skip the merging step, let me know if you would rather keep using it. And if we want to still use the merging step, would we want to include it in RSEC function arguments instead of separately?
EP: I don't think the merging step on the W makes a whole lot of sense – the method is irrelevant. The merging is based on calculating the % of genes found significant (the specific method is arbitrary). The best thing would be to replace the W with residuals in the assay of `ceObj` (or whatever data that you will do the

DE on for the time stuff below), and then run the merging step on that data. I'm not particularly fond of `locfdr`. It was probably the method that gave the best merging to Russell and Diya. You'd really have to run `mergeClusters` setting `plotInfo="all"` and look at the results and decide both the cutoff level and the method.
EP: Also, if you don't save the output of `mergeClusters` it doesn't update `ceObj`. I was calling it for just the resulting plots, since it was already merged in RSEC above. I've changed to code to update ceObj below.
FP: Ha ok, good to know. I'll keep the eval=FALSE for the moment.

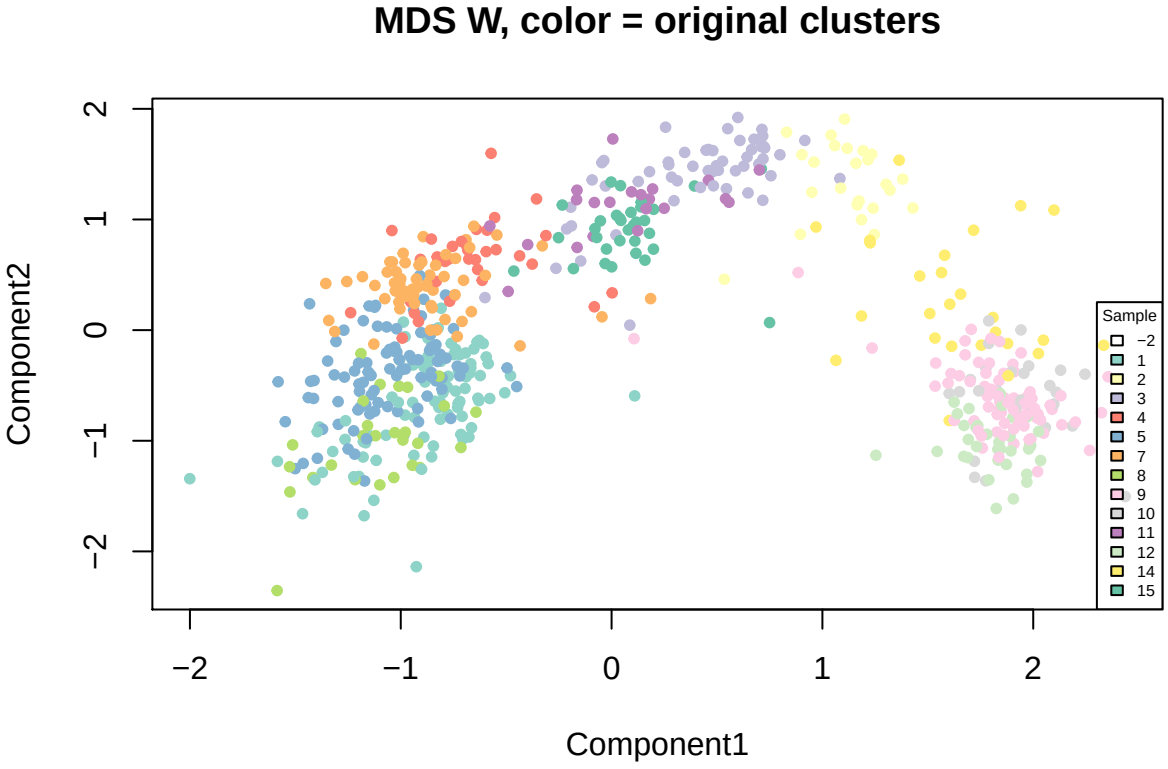
```
#re-does merging simpling to make plot
#something like:
#assay(ceObj)
# if that replacement data should be considered on the transformed scale in plots, etc, the transfor
#transformation(ceObj)
ceObj<-mergeClusters(ceObj, mergeMethod = "locfdr",
              plotInfo = "mergeMethod", cutoff = 0.01)
```

So, let's look at a heatmap on normalized values.
FP: Elizabeth, I did not find how to define the column annotation track in the plot below to have the same colors as in `ceObj@clusterLegend`[[1]]. I tried to use arguments annColors and annCol from aheatmap as it is said in plotHeatmap documentation that for signature matrix arguments can be passed to aheatmap. But I got the error "The following arguments to aheatmap cannot be set by the user in plotHeatmap:Rowv,Colv,color,annCol,annColors".
EP: Fanny, you would need to use the argument 'clusterLegend'. That argument takes either the format of aheatmap (list with each element a *named* vector of colors) or the format of the clusterExperiment object (i.e. list with each element a matrix with columns for `name` and `color`). So I think the following code will run, though it might need the list to have names...
But an easier fix to the code would be to set `visualizeData` option. I haven't tested this because I don't have the objects need run, so let me know if there is error.
FP: it seems great to me, what do you think?
EP: We should be careful, because the default in plotHeatmap is to plot the 500 most variable genes (maybe a slightly paternalistic default). I've changed it to `all` in the code here. I've also added the plotting of the batch, experiment, and Russell's original clusters. We may not want to keep all of them, but probably at least Russell's clusters for comparison.

```
# sampleData <- data.frame(ours = primaryCluster(ceObj))
# plotHeatmap(assays(se)$normalizedValues,
#             main = 'Normalized values, 1000 most variable genes',
#             clusterSamplesData = ceObj@dendro_samples,
#             sampleData = as.matrix(sampleData),clusterLegend=ceObj@clusterLegend[1])
# easier fix:
colData(ceObj)$clusterLabels <- as.factor(colData(ceObj)$clusterLabels)
origClusterColors<-bigPalette[1:nlevels(colData(ceObj)$clusterLabels)]
experimentColors<-bigPalette[1:nlevels(colData(ceObj)$Experiment)]
batchColors<-bigPalette[1:nlevels(colData(ceObj)$Batch)]
metaColors<-list("Experiment"=experimentColors,"Batch"=batchColors,"clusterLabels"=origClusterColors

plotHeatmap(ceObj, visualizeData = assays(se)$normalizedValues,
            whichClusters = "primary",clusterFeaturesData="all",
            clusterSamplesData = "dendrogramValue",
            sampleData=c("clusterLabels","Batch","Experiment"),
            clusterLegend=metaColors, annLegend=FALSE,
            main = 'Normalized values, 1000 most variable genes',
            breaks = 0.99)
```

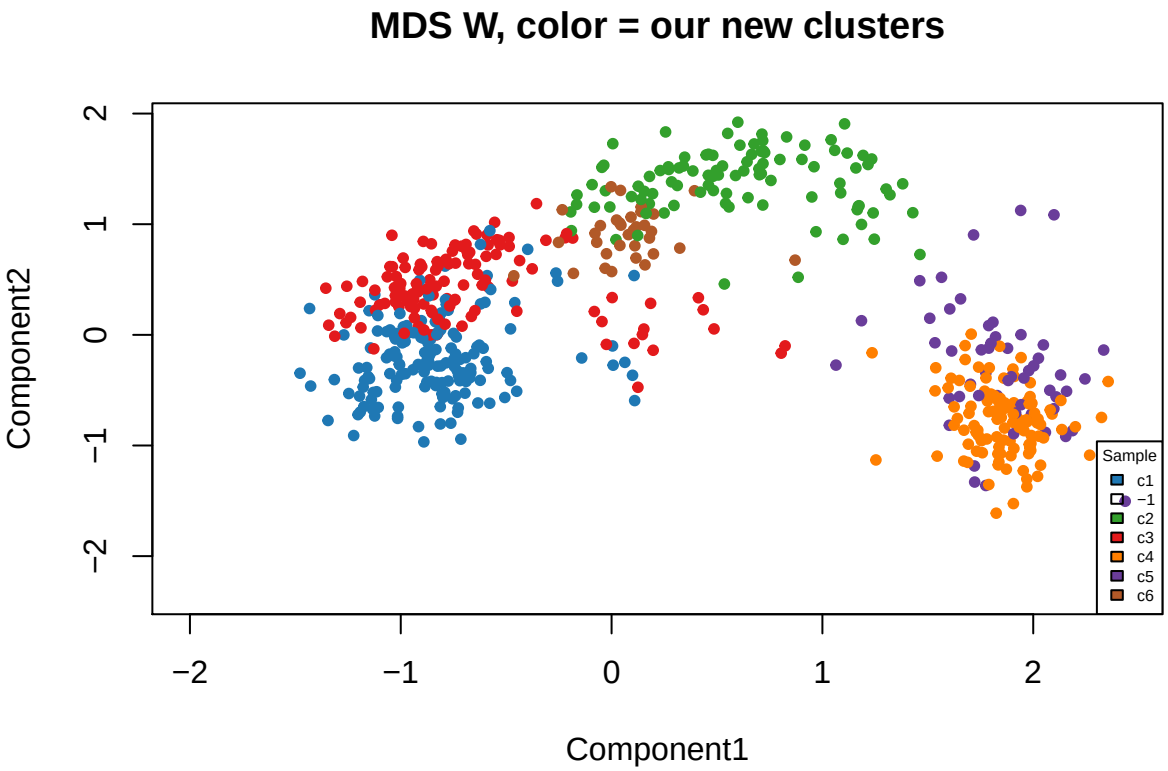



```
plot(fit$points, col = col_clus[as.character(clus.labels)],
     main = 'MDS W, color = original clusters', pch = 20,
     xlab = 'Component1', ylab = 'Component2')
legend(x = 'bottomright', legend = unique(names(col_clus)), cex = .5,
       fill = unique(col_clus), title = 'Sample')
```




```
palDF <- ceObj@clusterLegend[[1]]
pal <- palDF[, 'color']
names(pal) <- palDF[, 'name']
pal["-1"] = "transparent"
plot(fit$points, col = pal[primaryClusterNamed(ceObj)],
     main = 'MDS W, color = our new clusters', pch = 20,
```

```
     xlab = 'Component1', ylab = 'Component2')
legend(x = 'bottomright', legend = names(pal), cex = .5,
       fill = pal, title = 'Sample')
```



Pseudotime ordering

The goal of this section is to see if we need to refit zinbwave when we want to run slingshot. We first run slingshot on the W used by clusterExperiment. In the second part of this section, we fit zinbwave on the matrix of counts where the unassigned cells have been removed. For each part (without or with refitting zinbwave), we run slingshot in the supervised and unsupervised mode and try k=3, k=4, k=5 dimensions in W.
From what I understand, start original clusters are 1 and 5 (HBC) and end original clusters are 15 (Microvillus), 9 and 12 (neuron), and 4, 7 (Sus). Additionally, we want the GBC cluster to be a junction before the differentiation between Microvillus and Neuron. The correspondance with the original clusters is as follow

```
table(data.frame(original = clus.labels, ours = primaryClusterNamed(ceObj)))
```

```
##          ours
## original -1 c1 c2 c3 c4 c5 c6
##       -2 51 50  6 33  5  3  3
##       1  41 49  0  0  0  0  0
##       2   1  0 24  0  0  0  0
##       3   2  2 49  1  0  0  0
##       4   1  2  0 32  0  0  0
##       5  36 55  0  2  0  0  0
##       7   3  1  0 54  0  0  0
##       8  27  0  0  0  0  0  0
##       9   2  0  1  1 68  2  0
##       10  0  0  0  0  0 26  0
##       11  3  2 16  0  0  0  0
##       12  0  0  0  0 35  0  0
##       14  4  0  1  0  0 21  0
##       15  1  0  1  0  0  0 30
```

```
Kvec <- c(3, 4, 5)
```

Cluster name	Description	Color	Correspondence
c1	HBC	blue	original 1, 5
c2	GBC	immature neurons	MV 1
c3	Sus	red	original 4, 7
c4	Neuron	orange	original 9, 12
c5	Immature Neuron	purple	original 10, 14
c6	Microvillus	cyan	original 15

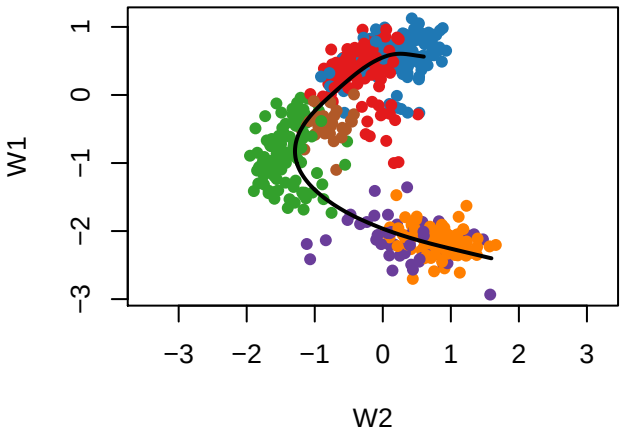
Use W components

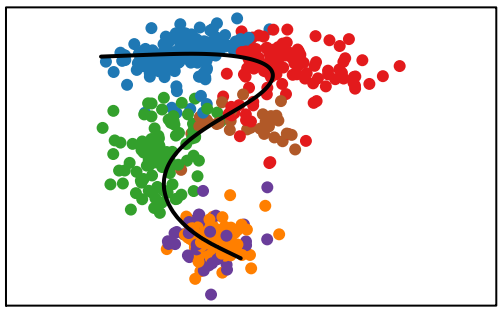
Use previous W The input of slingshot is the W used for clusterExperiment where the number of dimensions is reduced to k where k in (3, 4, 5) here.

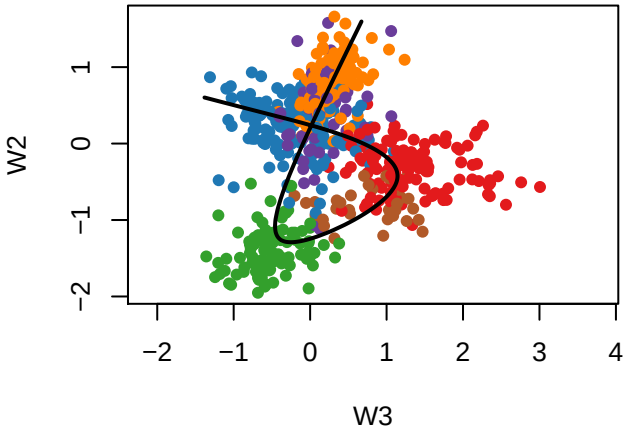
Unsupervised K = 3 only one lineage: sus is right after HBC
K = 4 same as K=3
K = 5 three lineages: hbc-gbc-immature-neuron; hbc-gbc-mv; hbc-sus (perfect!)

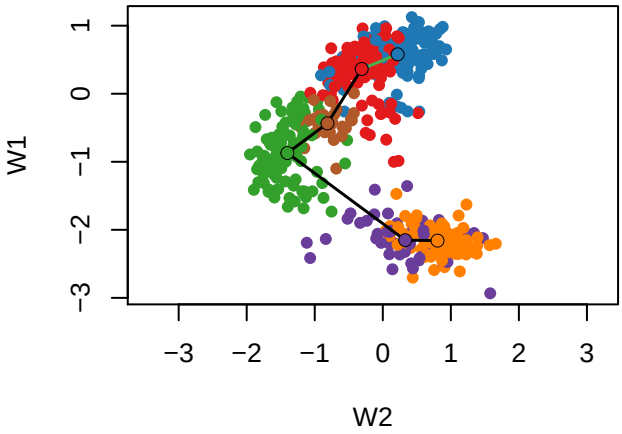
```
our_cl <- primaryClusterNamed(ceObj)
cl = our_cl[our_cl != "-1"]
pal = pal[names(pal) != '-1']
```

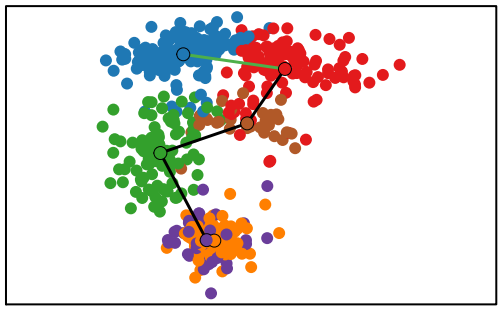
```
for (k in Kvec){
  X <- W[our_cl != "-1", 1:k]

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1")
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
  print(lineages$lineage5)
}
```

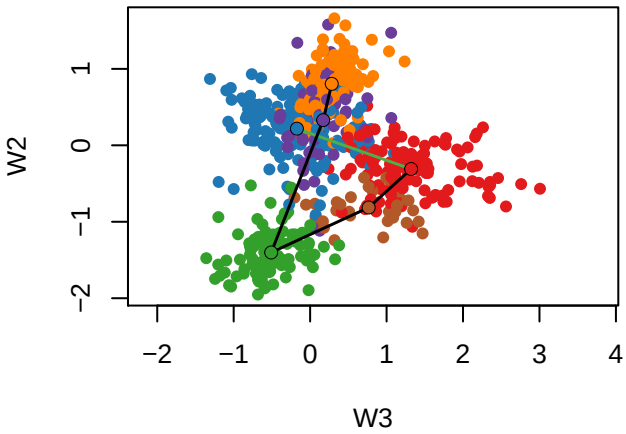






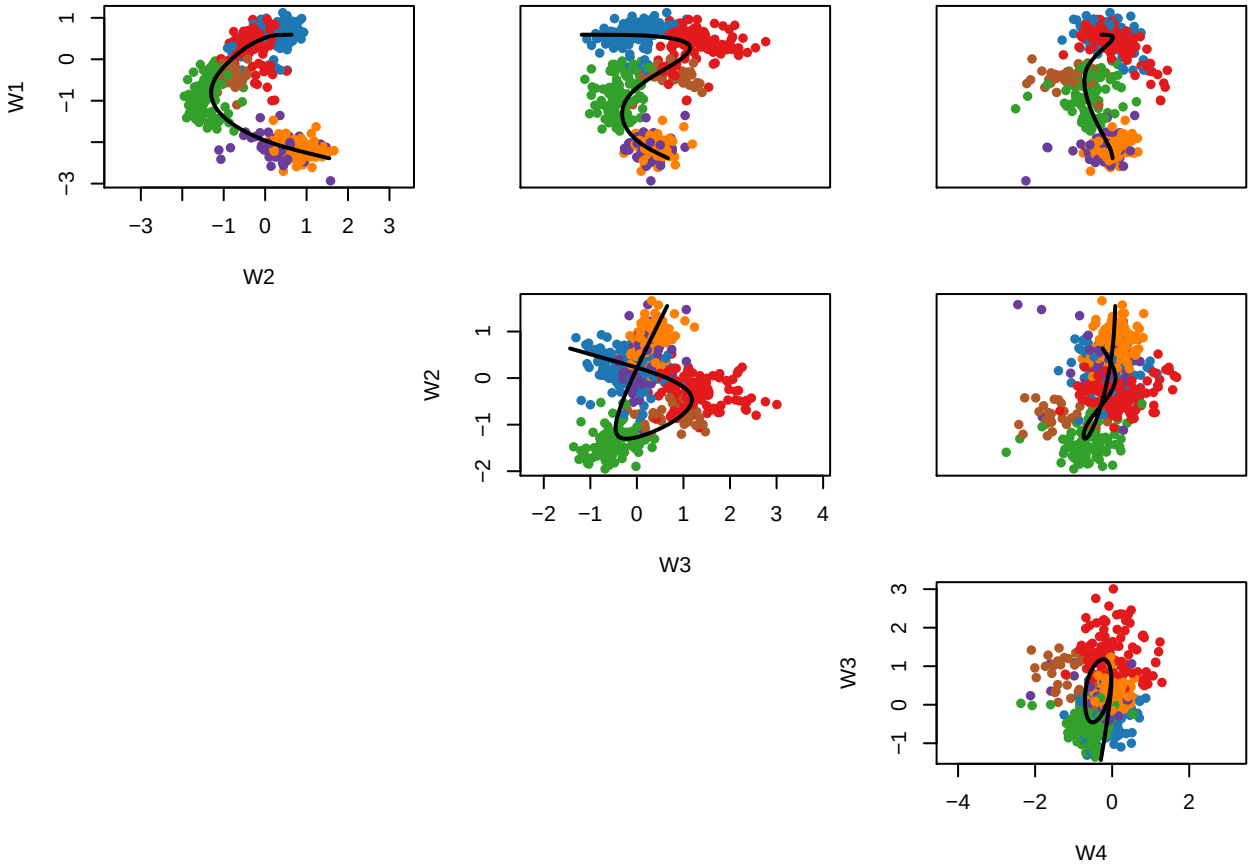


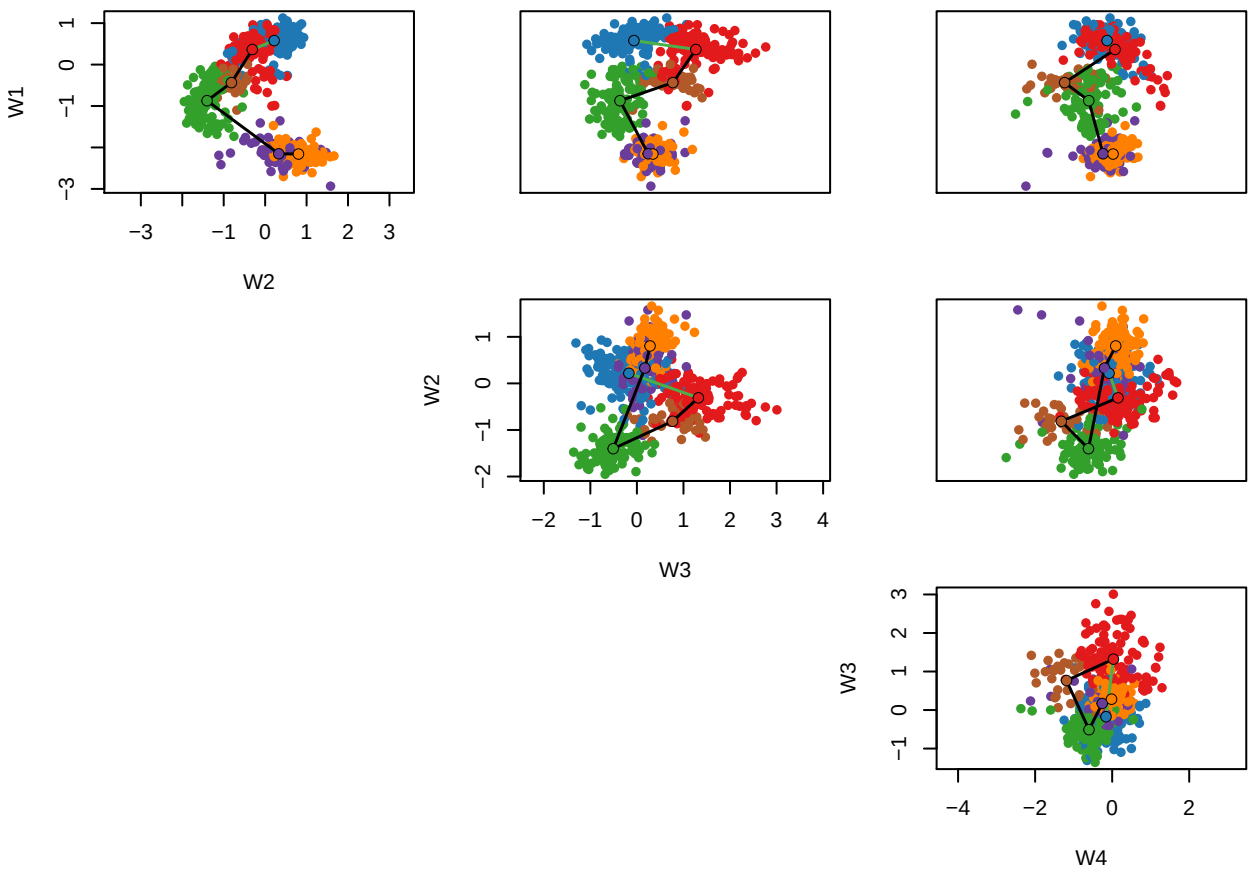





```
## [1] "K=3"
## [1] "c1" "c3" "c6" "c2" "c5" "c4"
## NULL
## NULL
## NULL
```

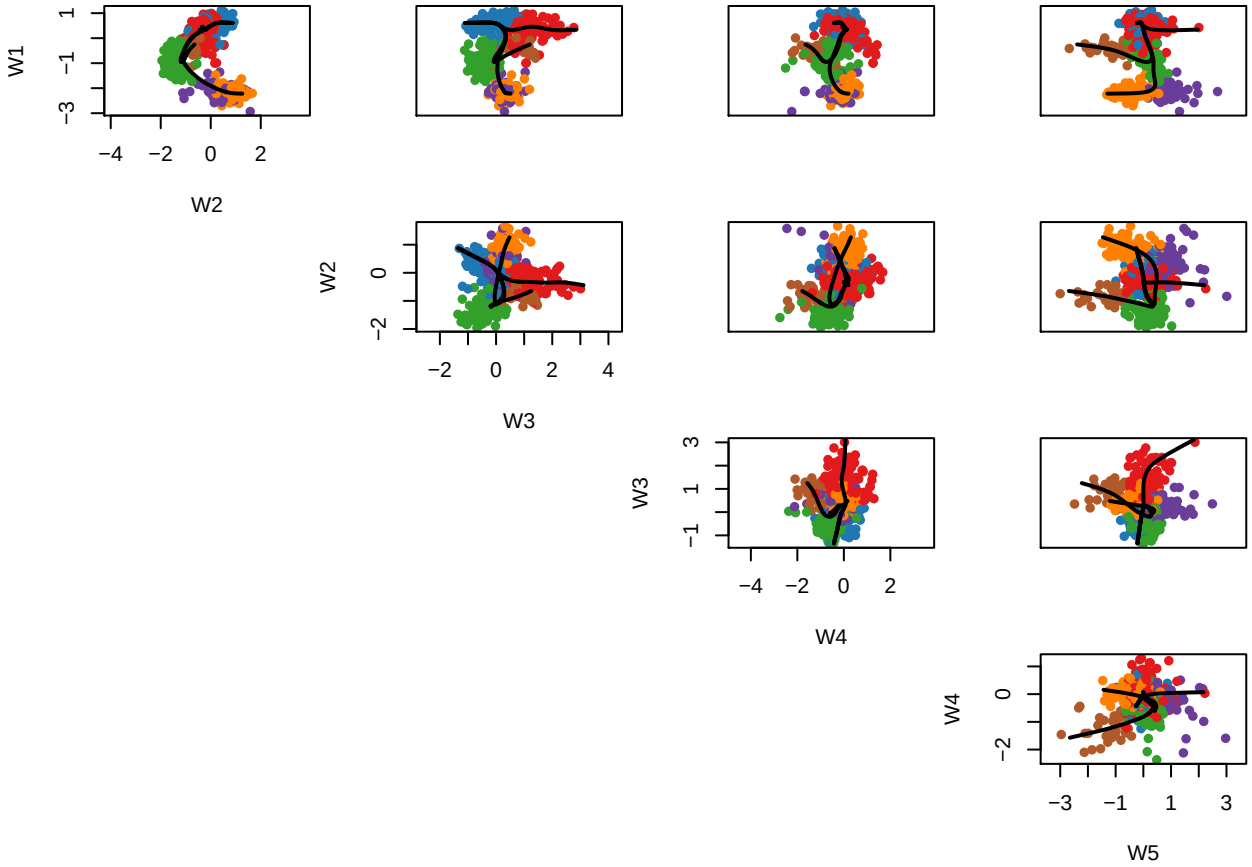
```
## NULL
```

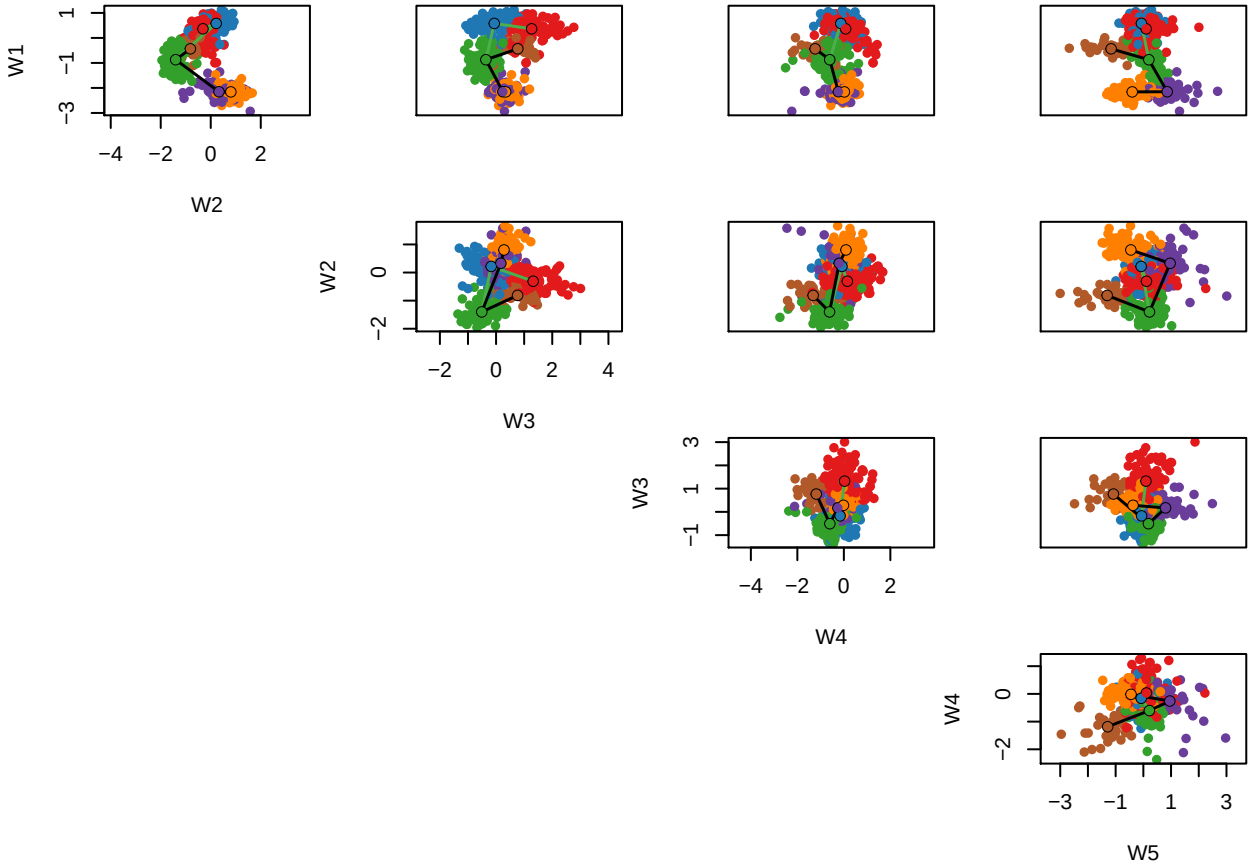





```
## [1] "K=4"
## [1] "c1" "c3" "c6" "c2" "c5" "c4"
```

```
## NULL
## NULL
## NULL
## NULL
```

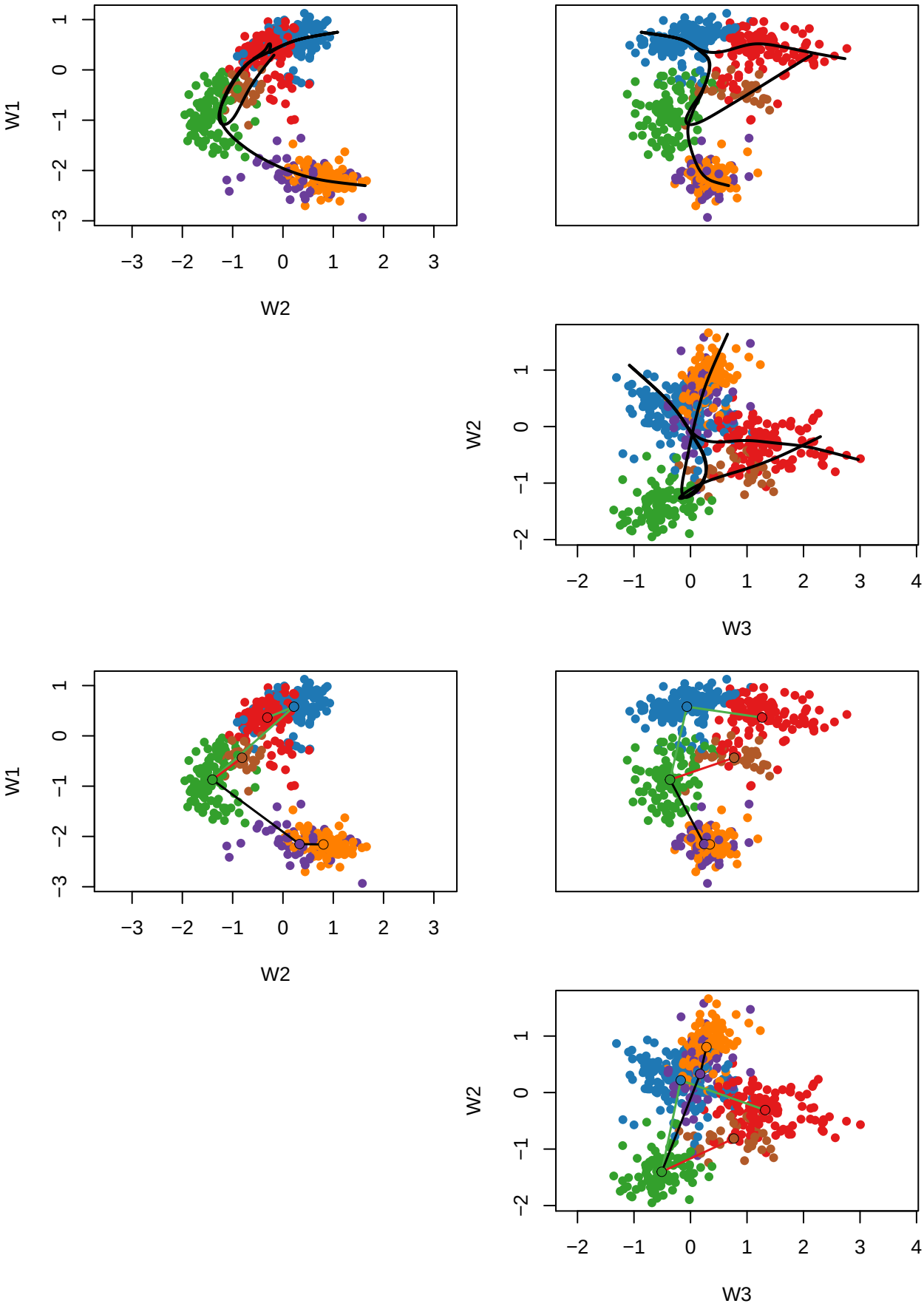





```
## [1] "K=5"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

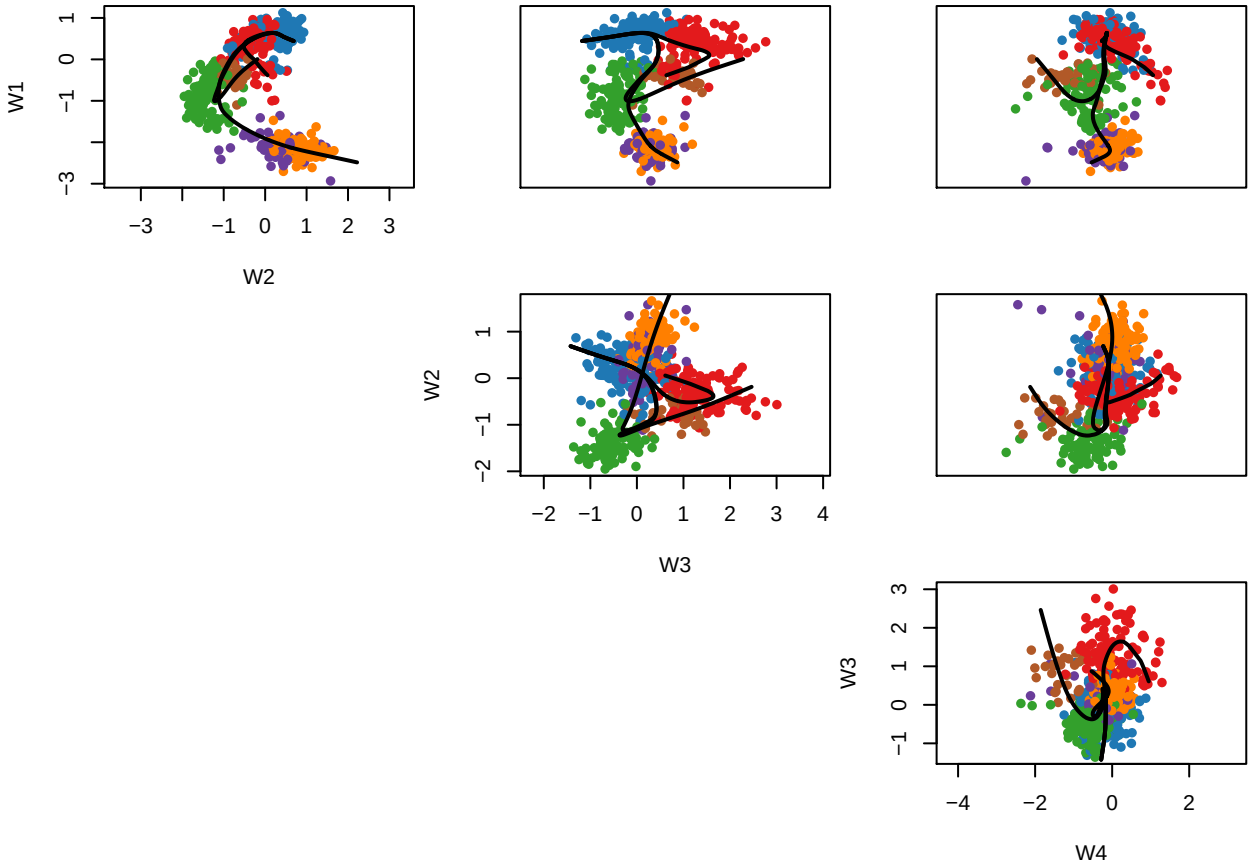
Supervised K = 3, 4, 5 get same results, all good.

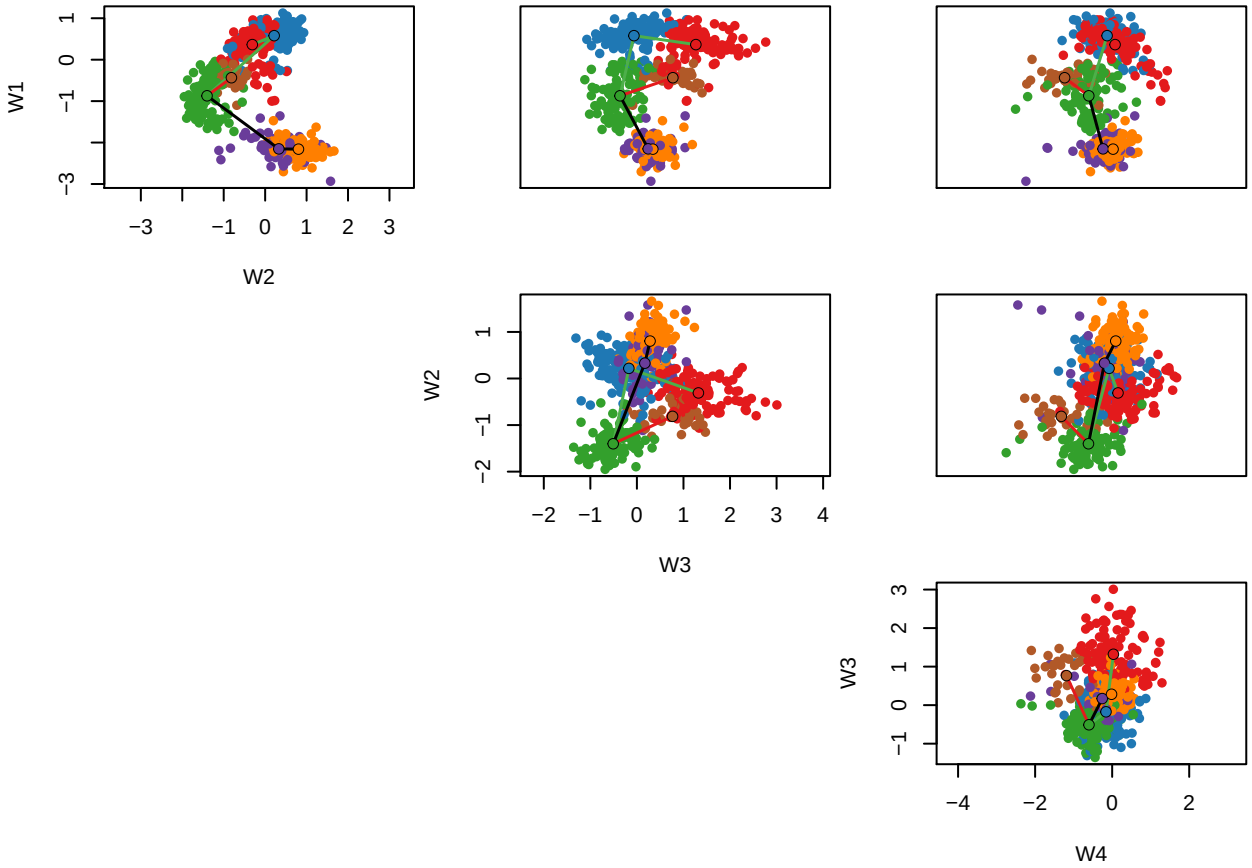
```
for (k in Kvec){
  X <- W[our_cl != "-1", 1:k]

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1",
                           end.clus = c("c3", "c6"))
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
  print(lineages$lineage5)
}
```




```
## [1] "K=3"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
```

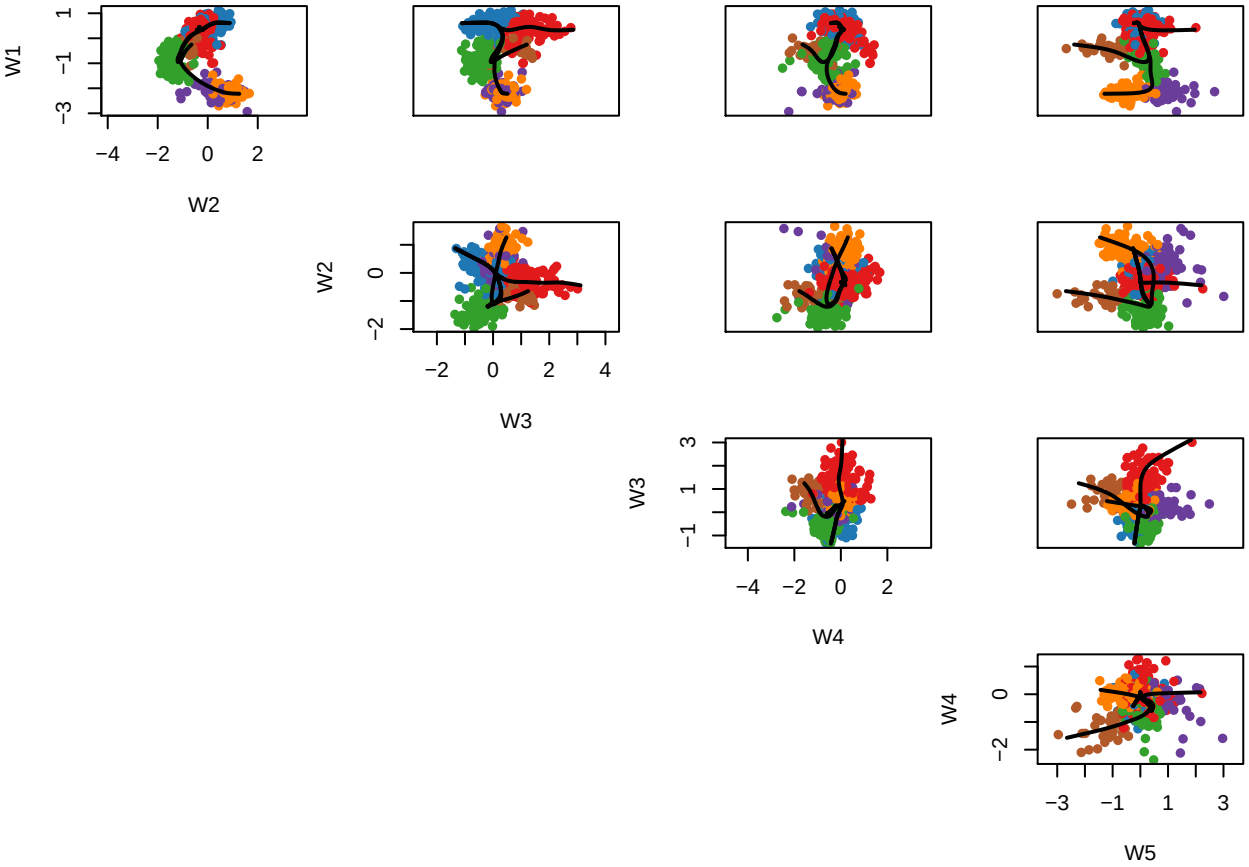
```
## NULL
## NULL
```

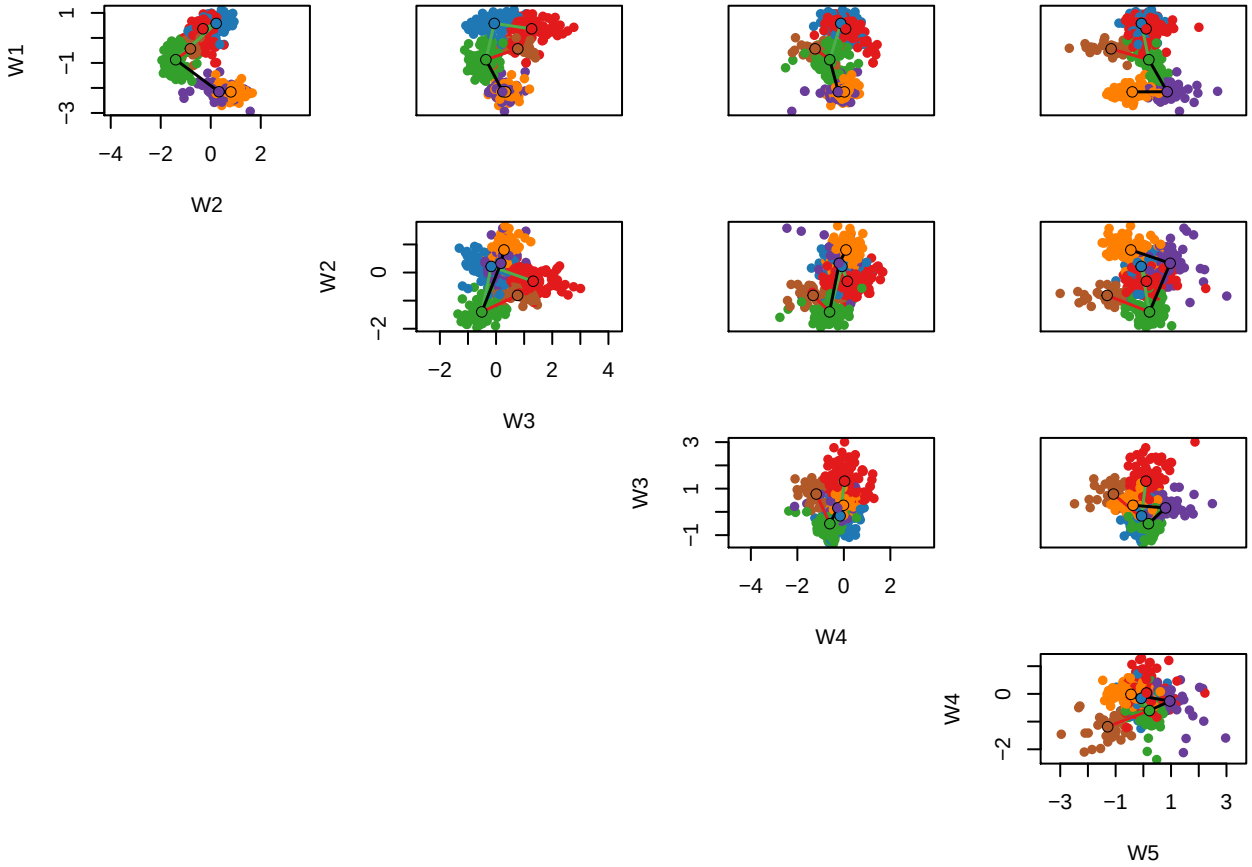





```
## [1] "K=4"
```

```
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```






```
## [1] "K=5"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

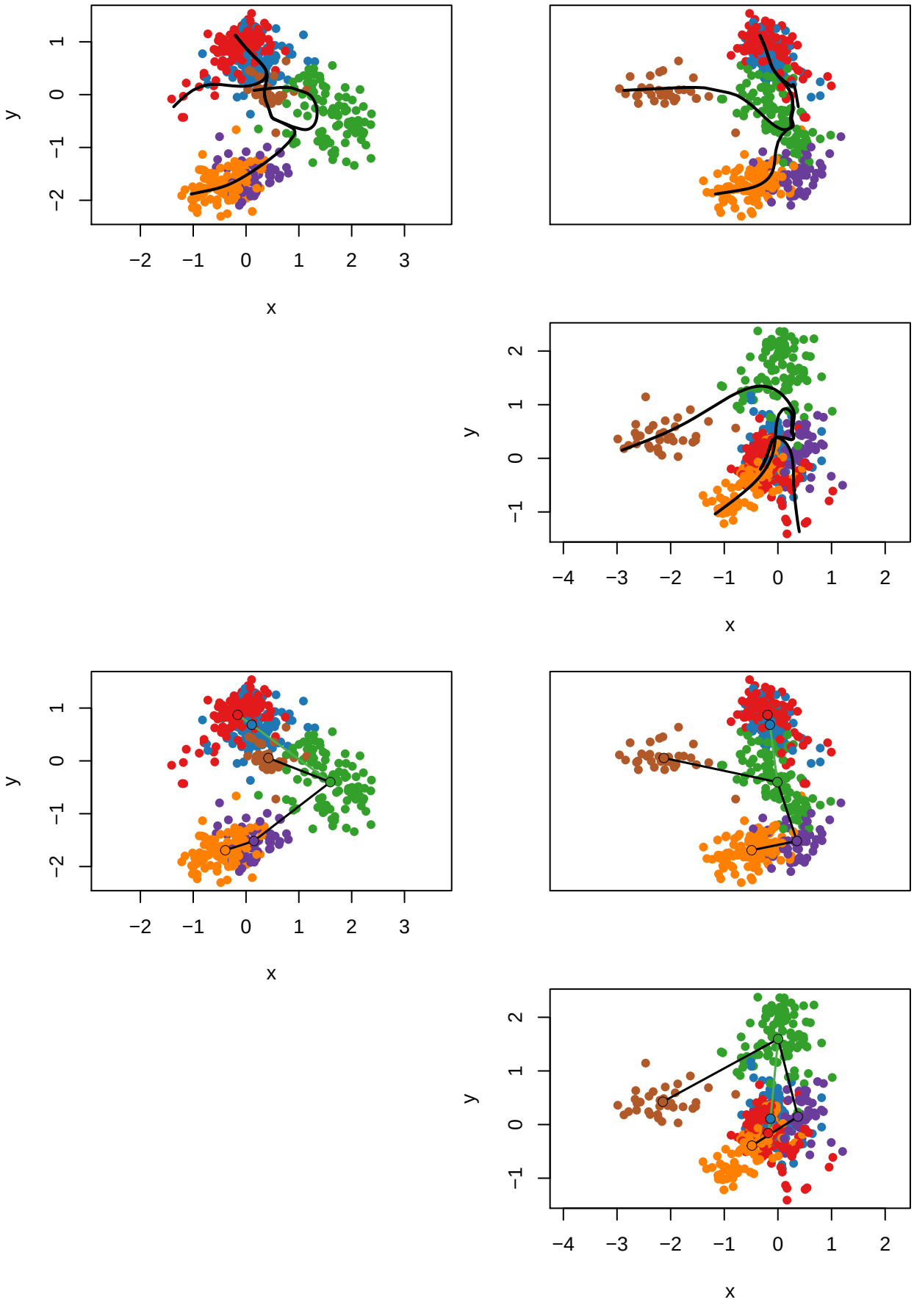
Re-fitting zinbwave

Unsupervised K = 3, 4, 5 get same results, all good.

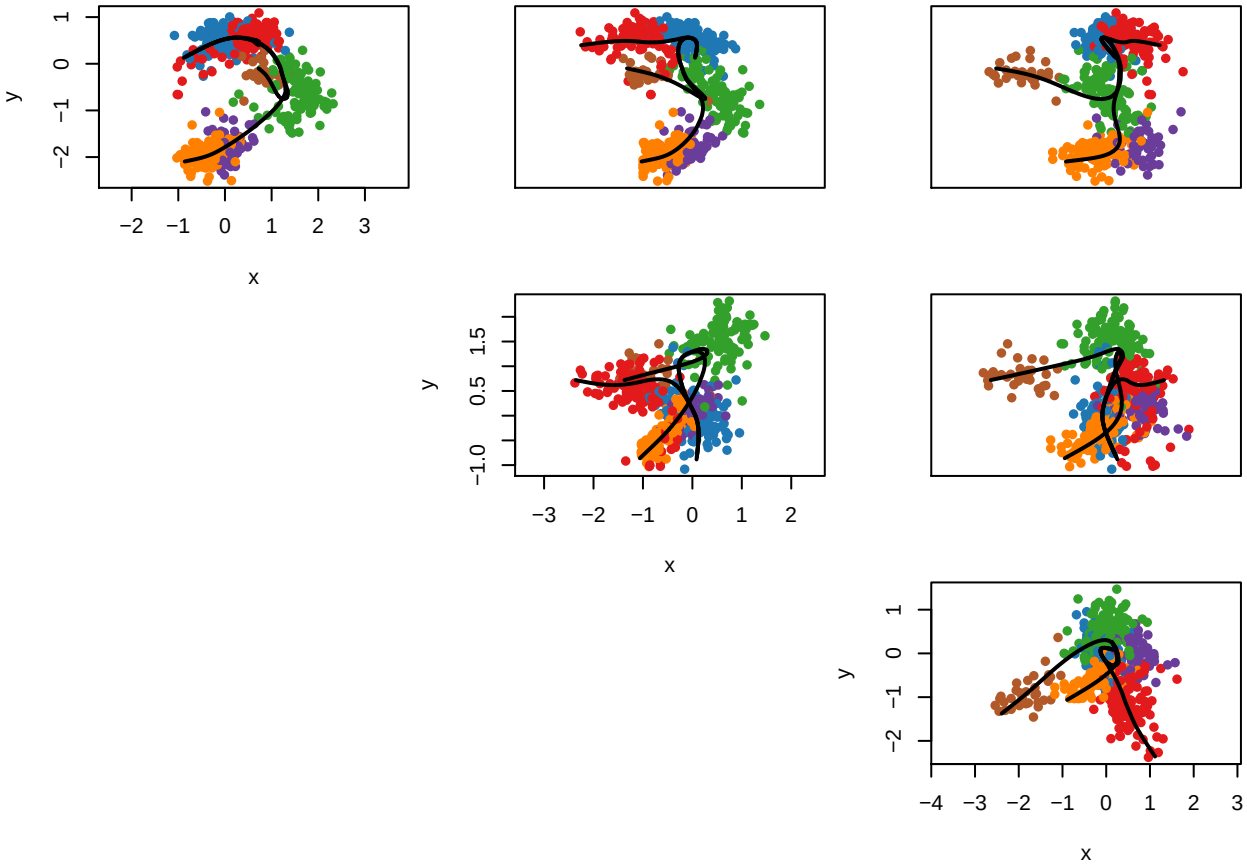
```
fn <- '../data/refit_zinbwave_slingshot_combineMinSize10.rda'

if (runZinb & !file.exists(fn)){
  zinbList <- lapply(Kvec, function(k){
    zinbFit(se[, our_cl != "-1"], X = '~ Batch', K = k)
  })
  save(zinbList, file = fn)
}else{
  load(fn)
}
```

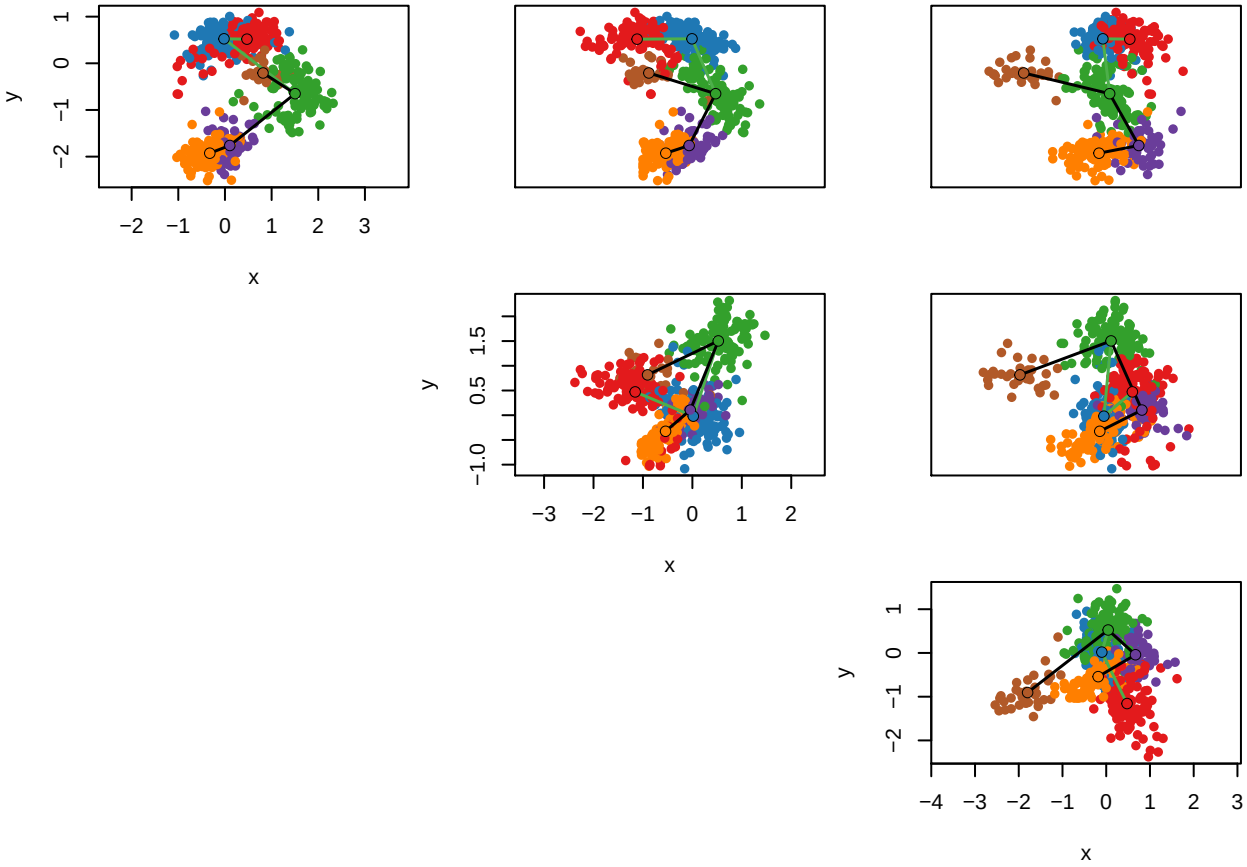
```
for(k in Kvec) {
  X <- getW(zinbList[[k - 2]])[, 1:k]

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1")
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
```

```
  print(lineages$lineage5)
}
```

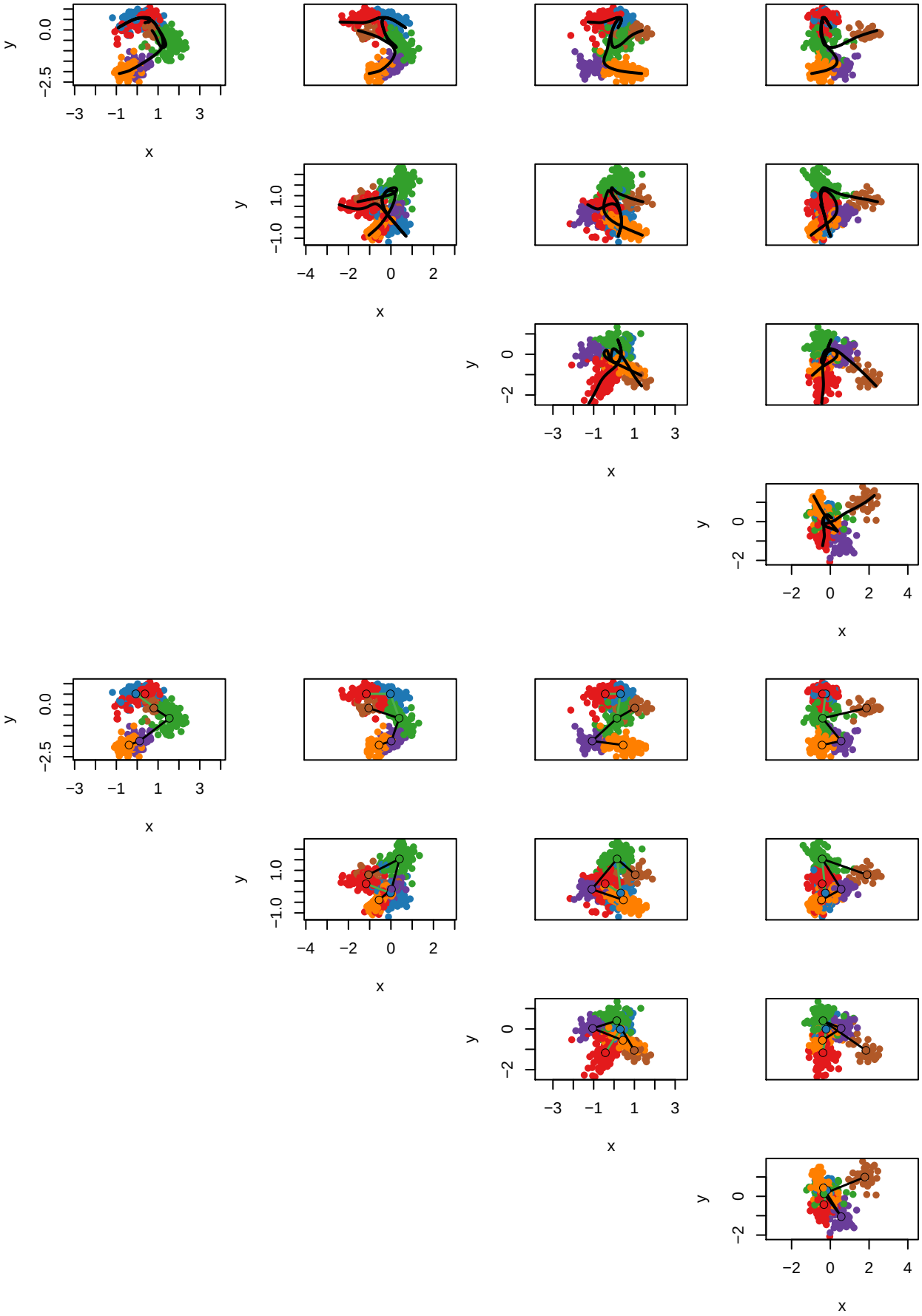


```
## [1] "K=3"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```






```
## [1] "K=4"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

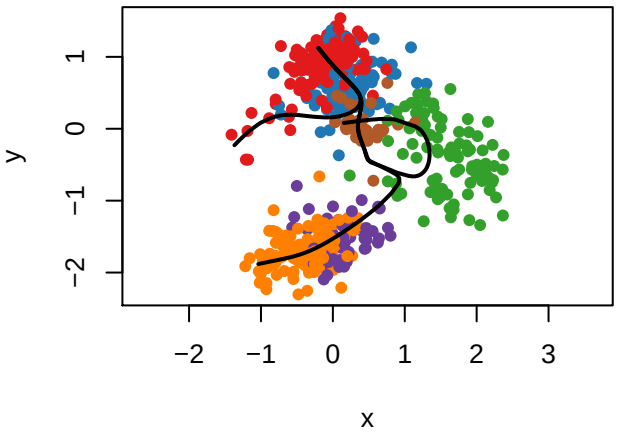


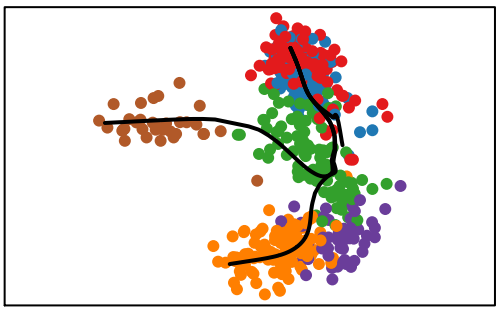

```
## [1] "K=5"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
```

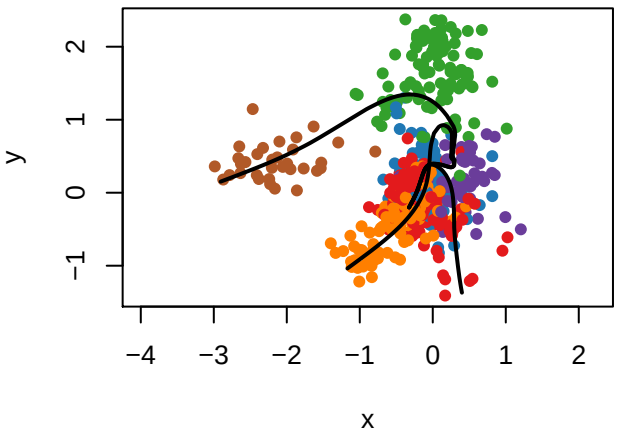
```
## NULL
## NULL
```

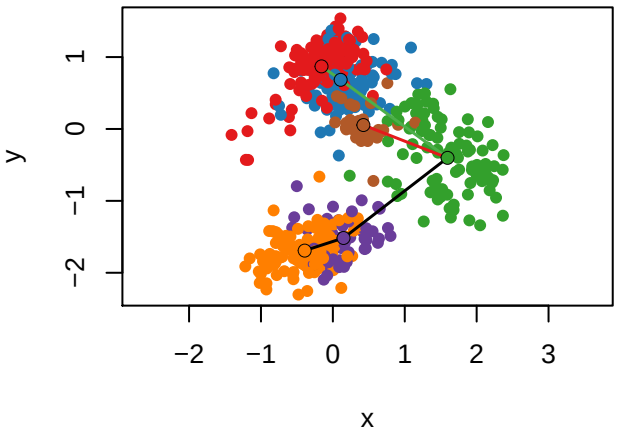
Supervised K = 3, 4, 5 get same results, all good.

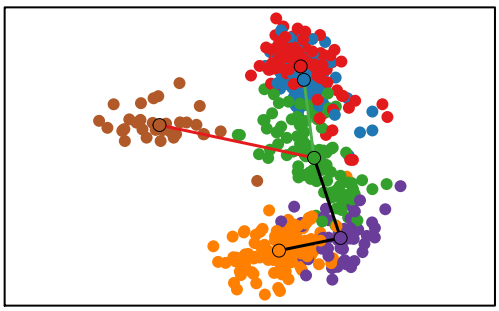
```
for(k in Kvec){
  X <- getW(zinbList[[k - 2]])[, 1:k]

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1",
                           end.clus = c("c3", "c6"))
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
  print(lineages$lineage5)
}
```

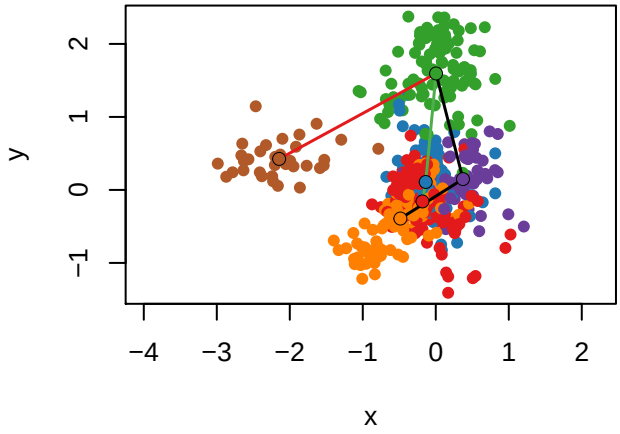




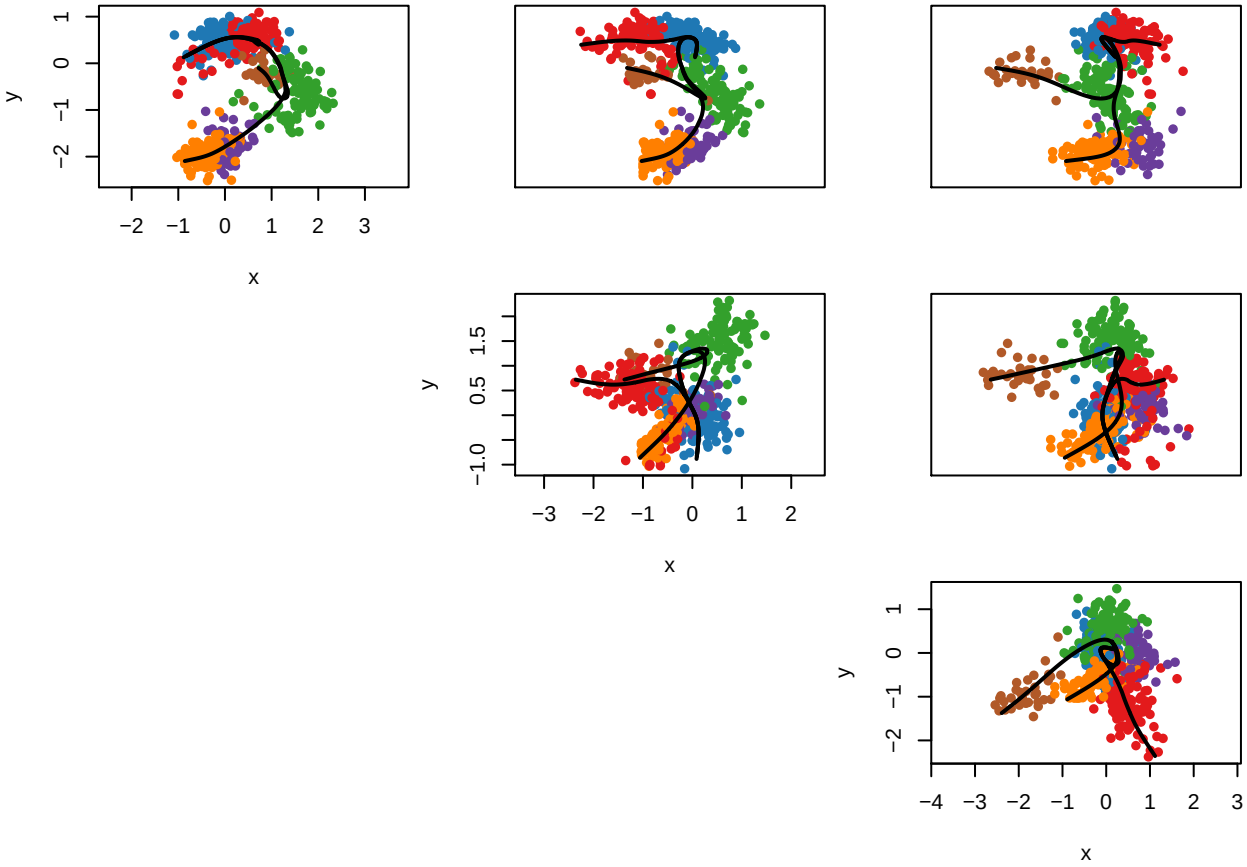


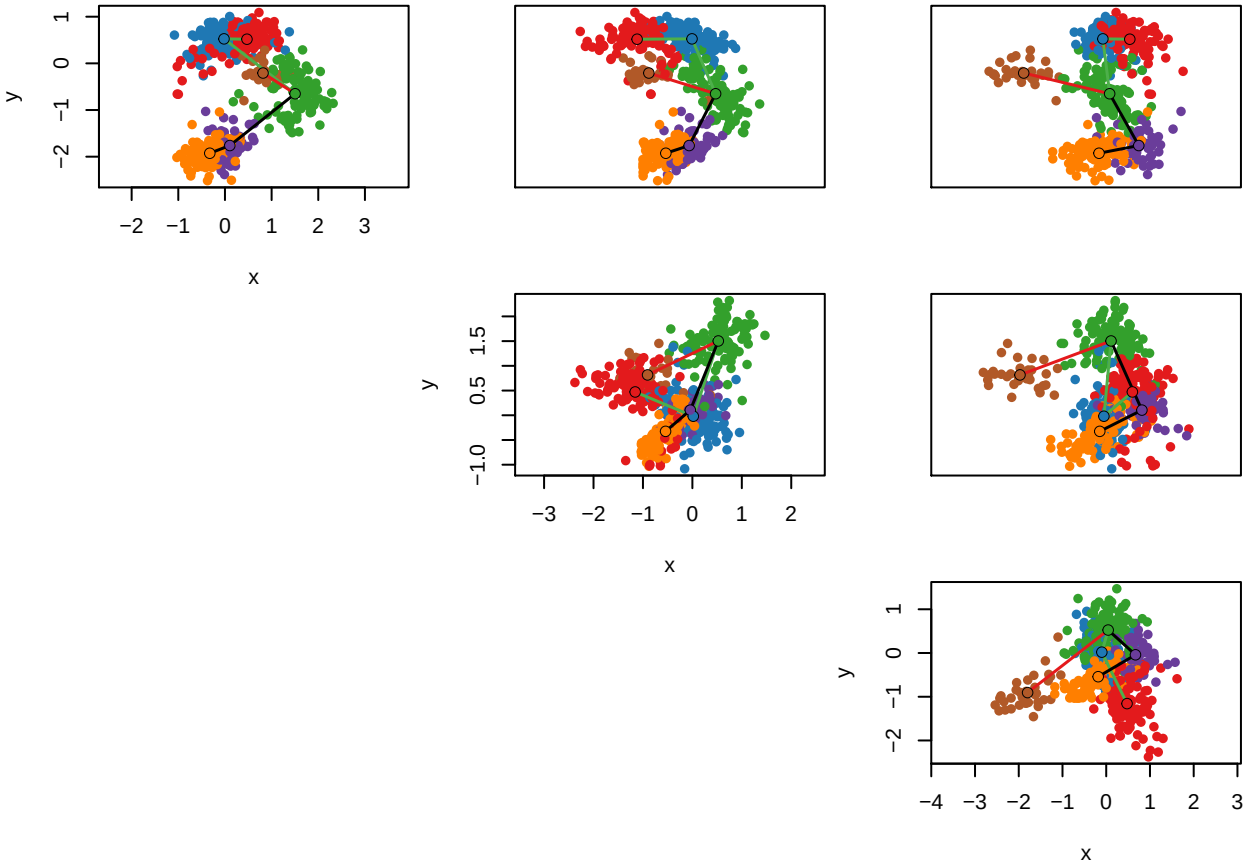






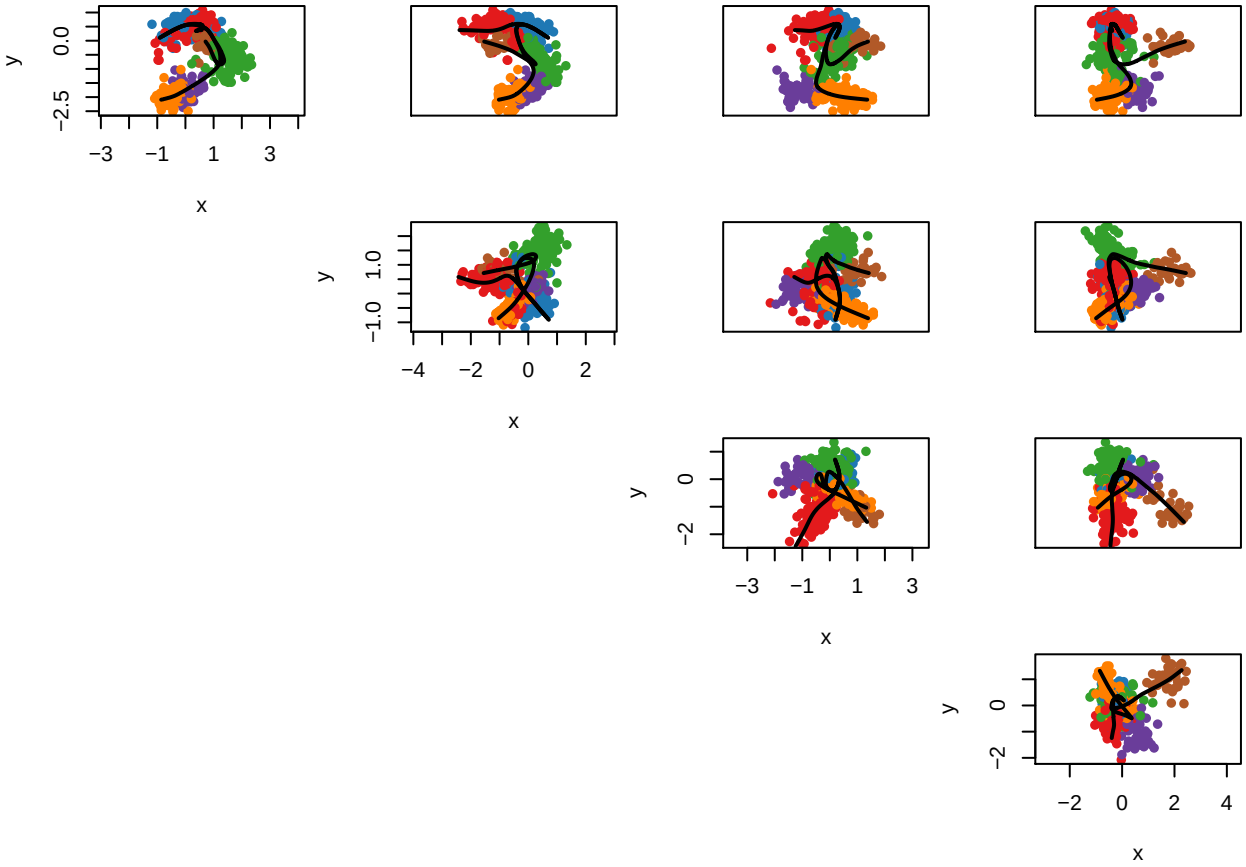

```
## [1] "K=3"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

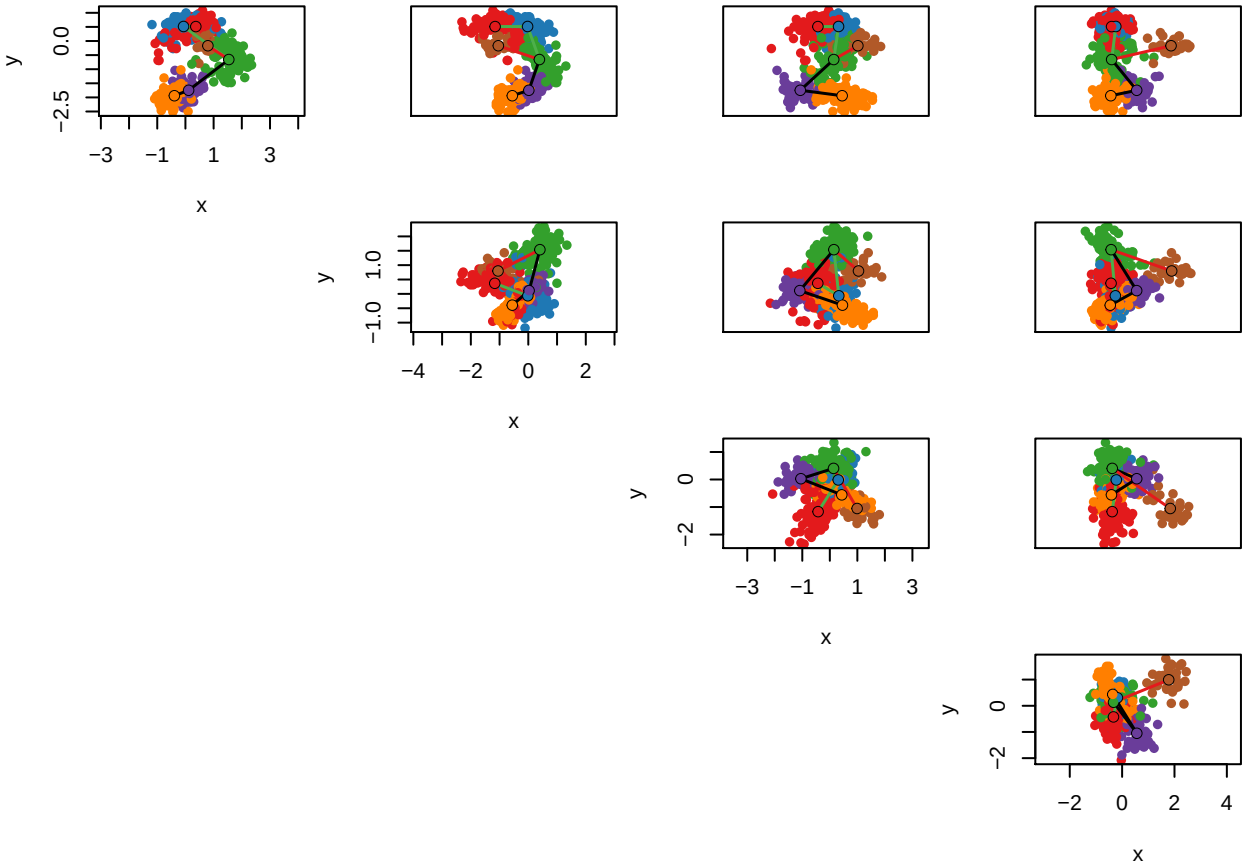





```
## [1] "K=4"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
```

```
## NULL
## NULL
```






```
## [1] "K=5"
```

```
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

Use MDS of W

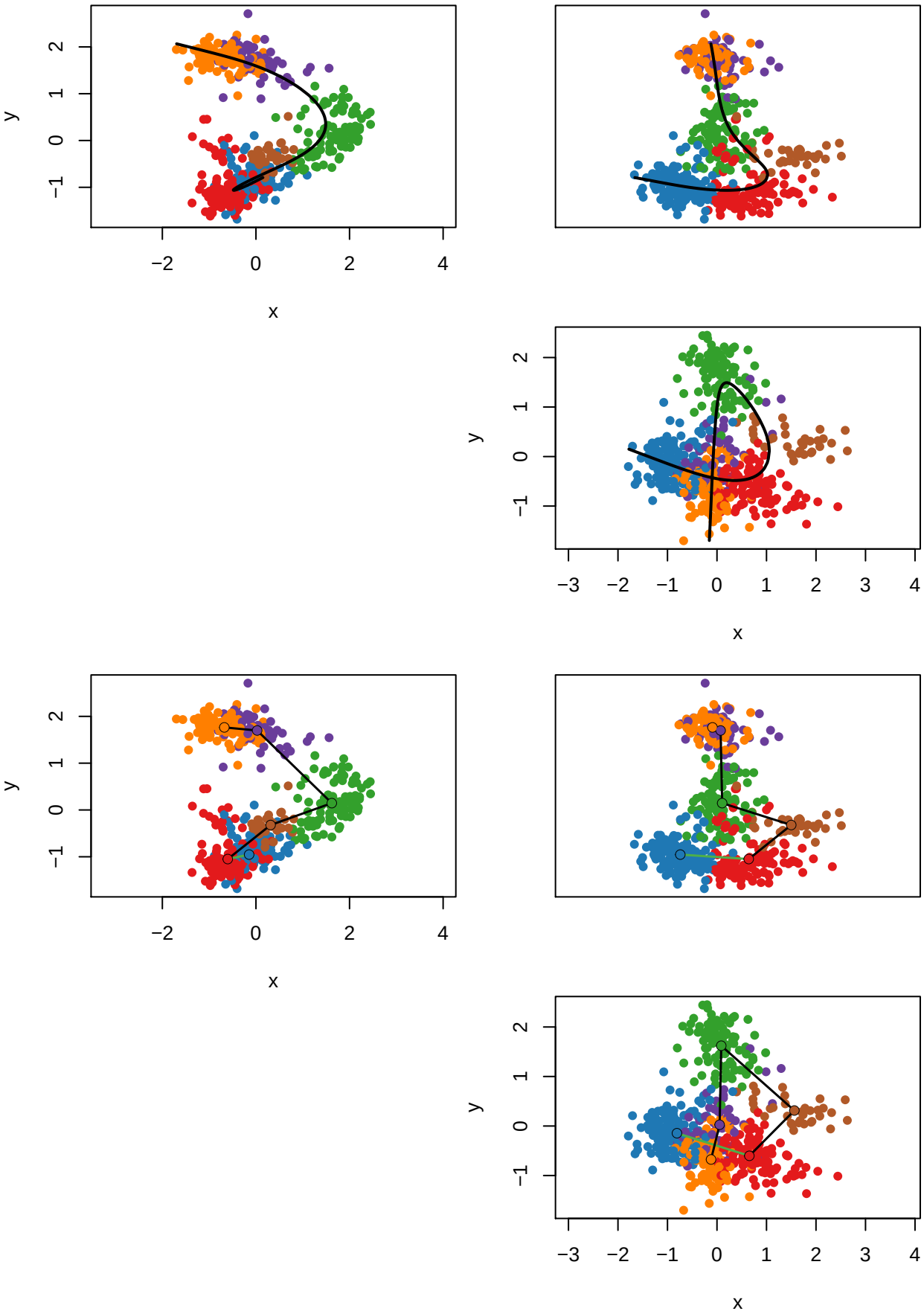
Here, instead of using the first components of W, we use the MDS of W.

Use previous W The input of slingshot is the MDS of W used for clusterExperiment where the number of dimensions is k where k in (3, 4, 5) here.

Unsupervised K = 3 only one lineage: sus is right after HBC
K = 4, 5 now, it is ok, three lineages: hbc-gbc-immature-neuron; hbc-gbc-mv; hbc-sus (perfect!)

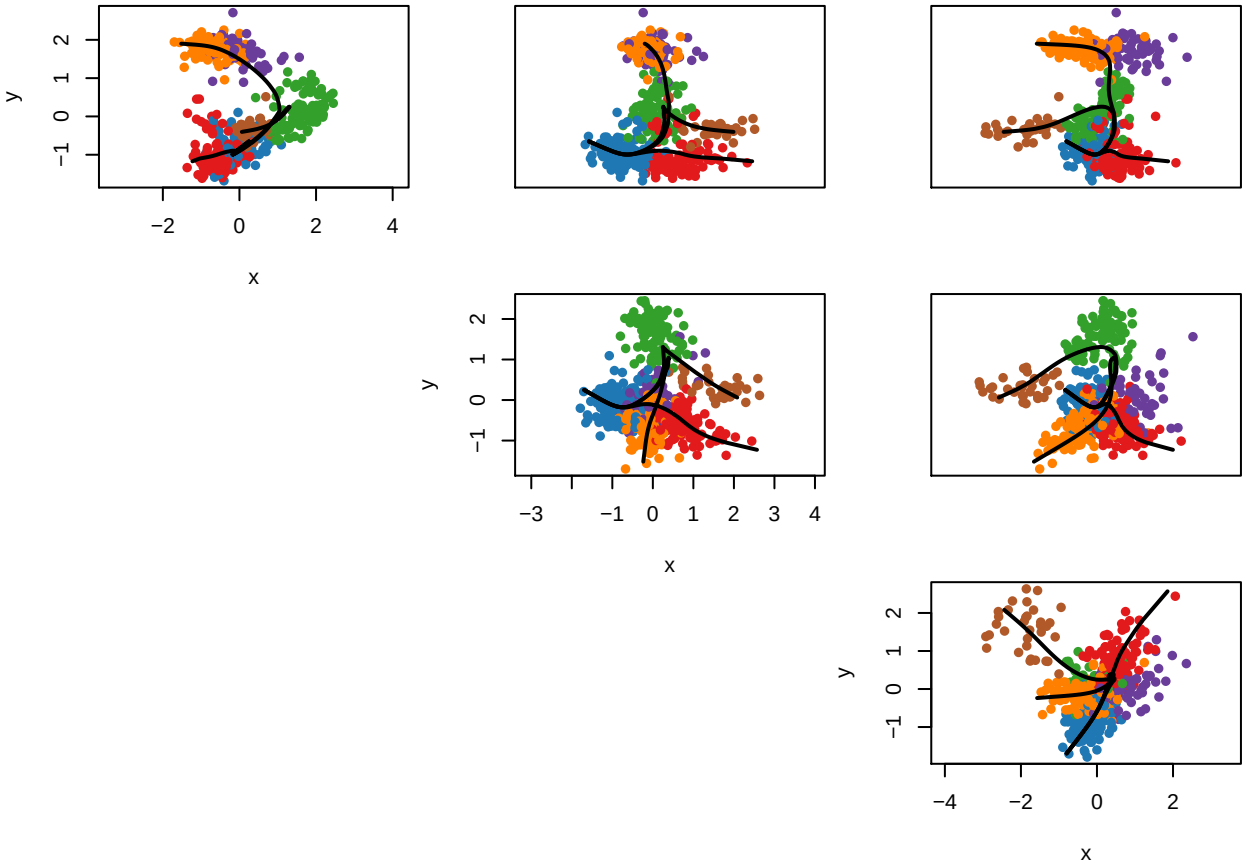
```
our_cl <- primaryClusterNamed(ceObj)
cl = our_cl[our_cl != "-1"]
pal = pal[names(pal) != '-1']
```

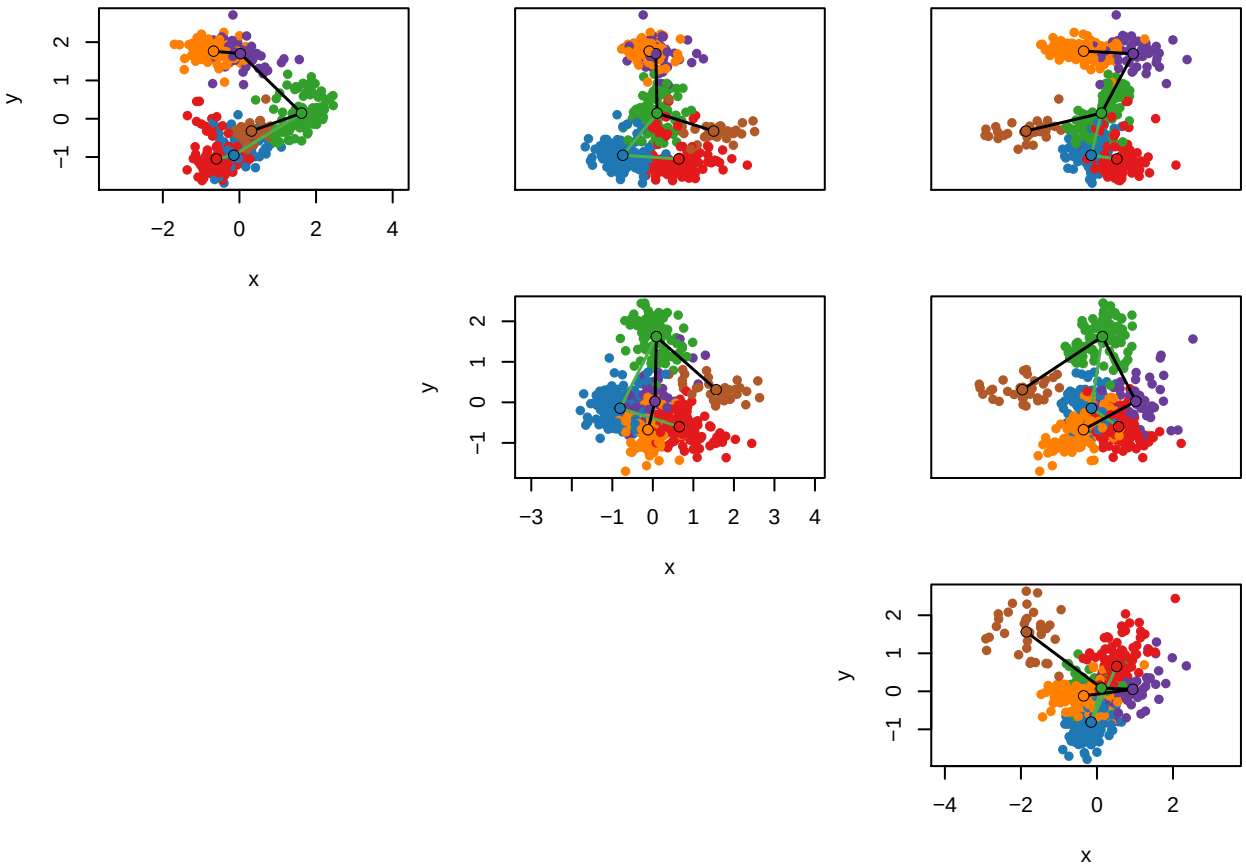
```
for (k in Kvec){
  X <- W[our_cl != "-1", ]
  mds <- cmdscale(dist(X), eig = TRUE, k = k)
  X <- mds$points

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1")
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
  print(lineages$lineage5)
}
```




```
## [1] "K=3"
## [1] "c1" "c3" "c6" "c2" "c5" "c4"
## NULL
## NULL
## NULL
```

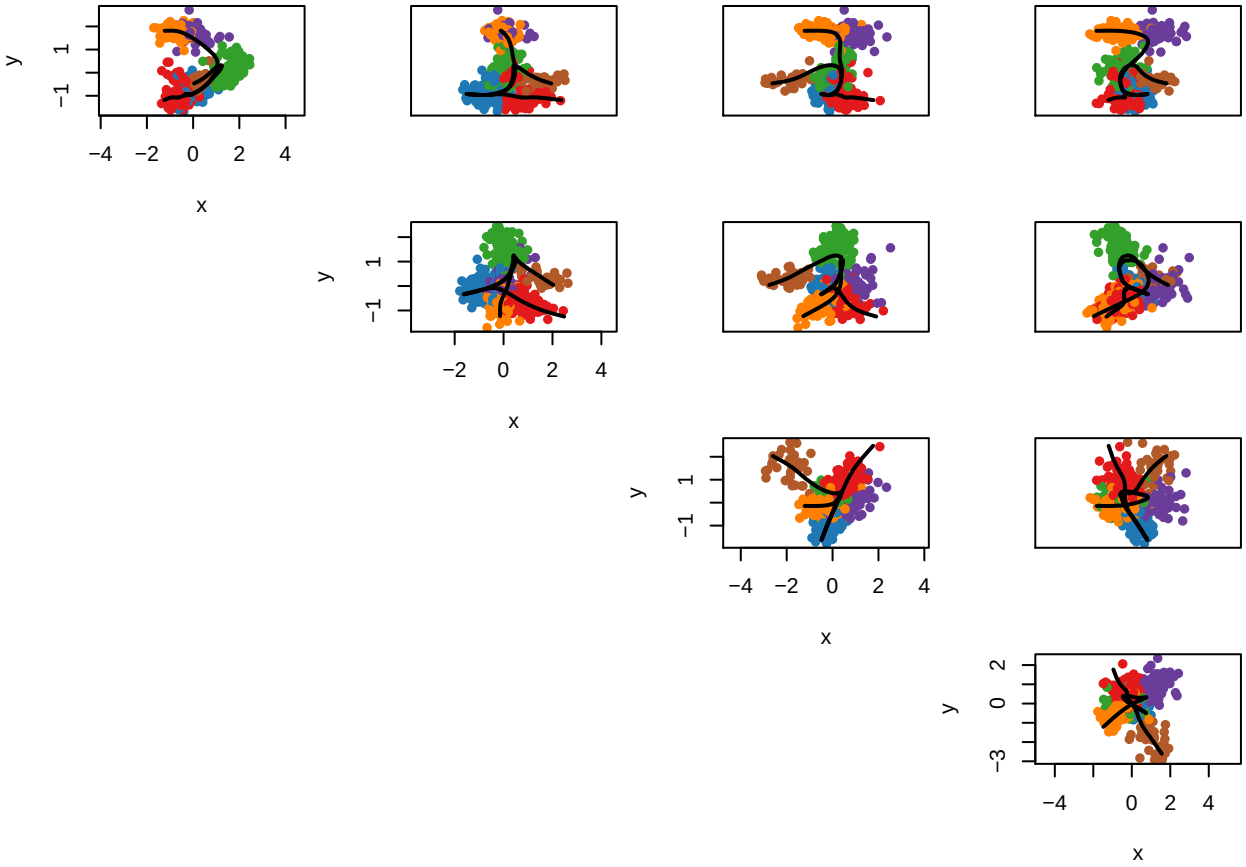
```
## NULL
```

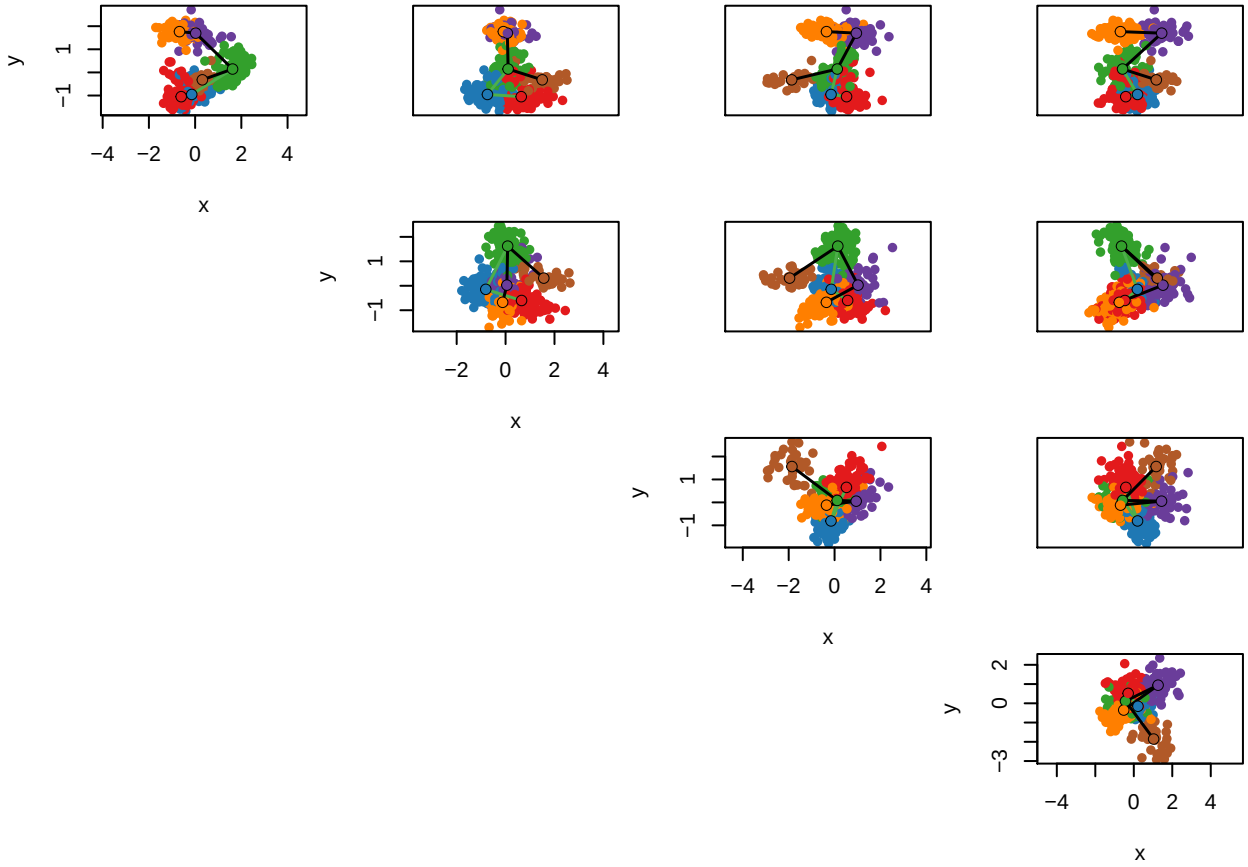





```
## [1] "K=4"
## [1] "c1" "c2" "c5" "c4"
```

```
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

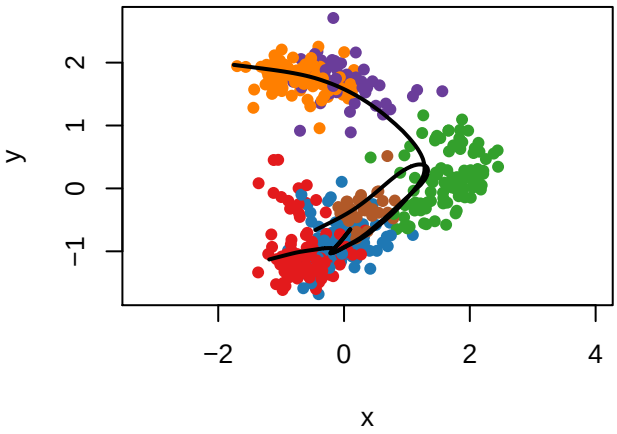


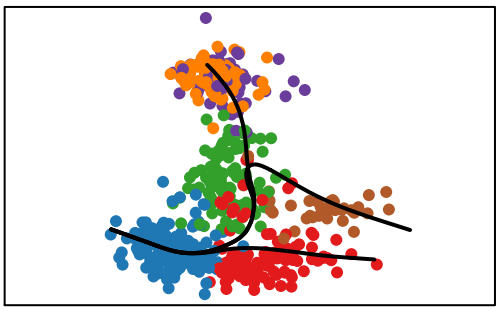


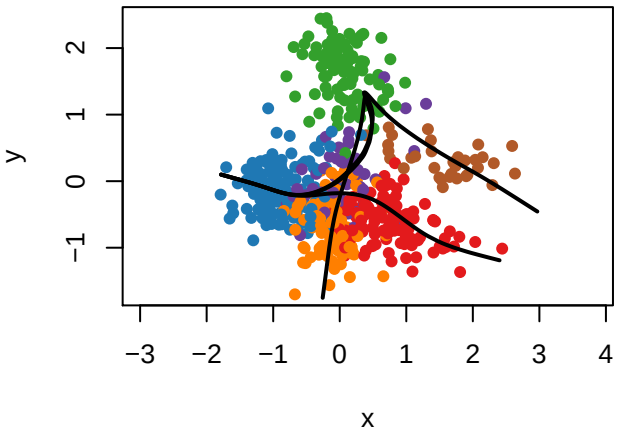

```
## [1] "K=5"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

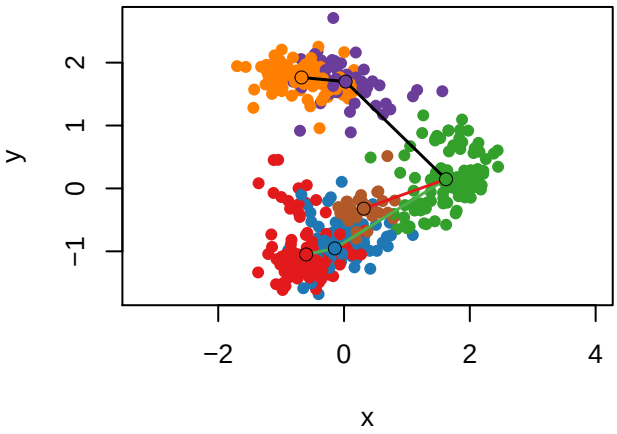
Supervised K = 3, 4, 5 get same results, all good.

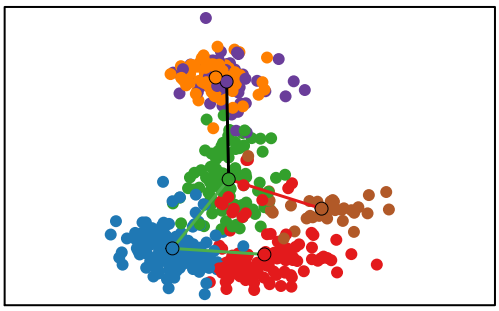
```
for (k in Kvec){
  X <- W[our_cl != "-1", ]
  mds <- cmdscale(dist(X), eig = TRUE, k = k)
  X <- mds$points

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1",
                           end.clus = c("c3", "c6"))
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
  print(lineages$lineage5)
}
```

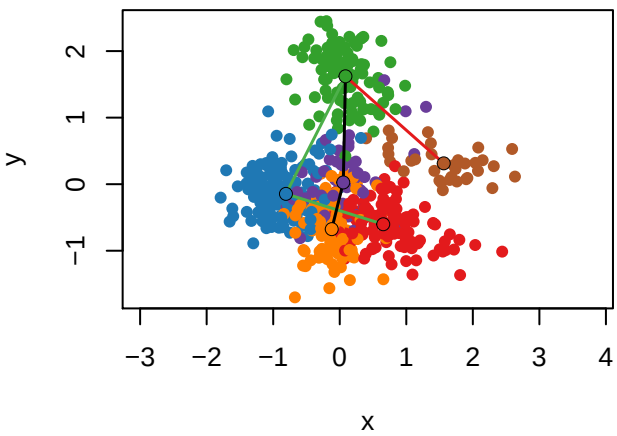






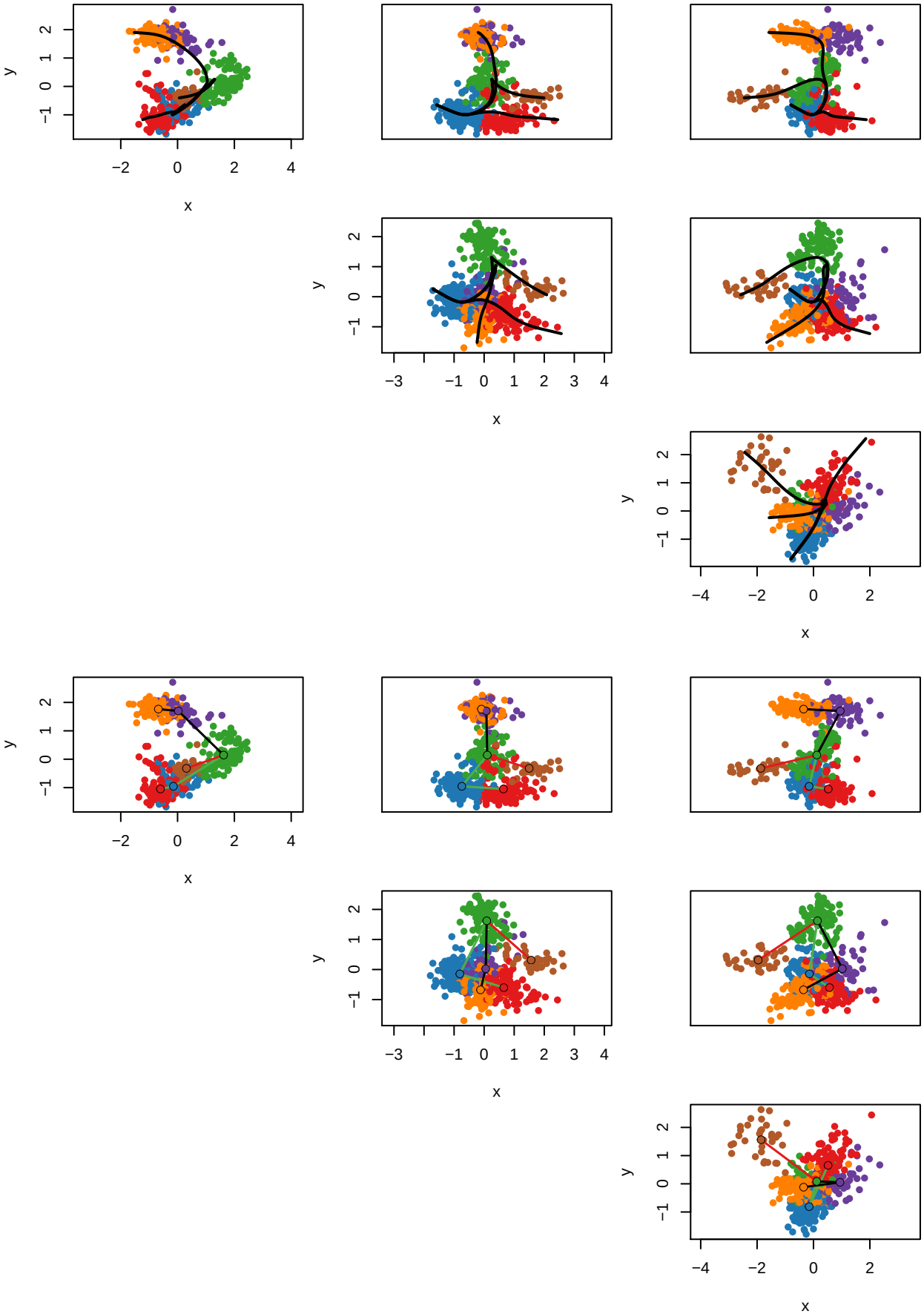






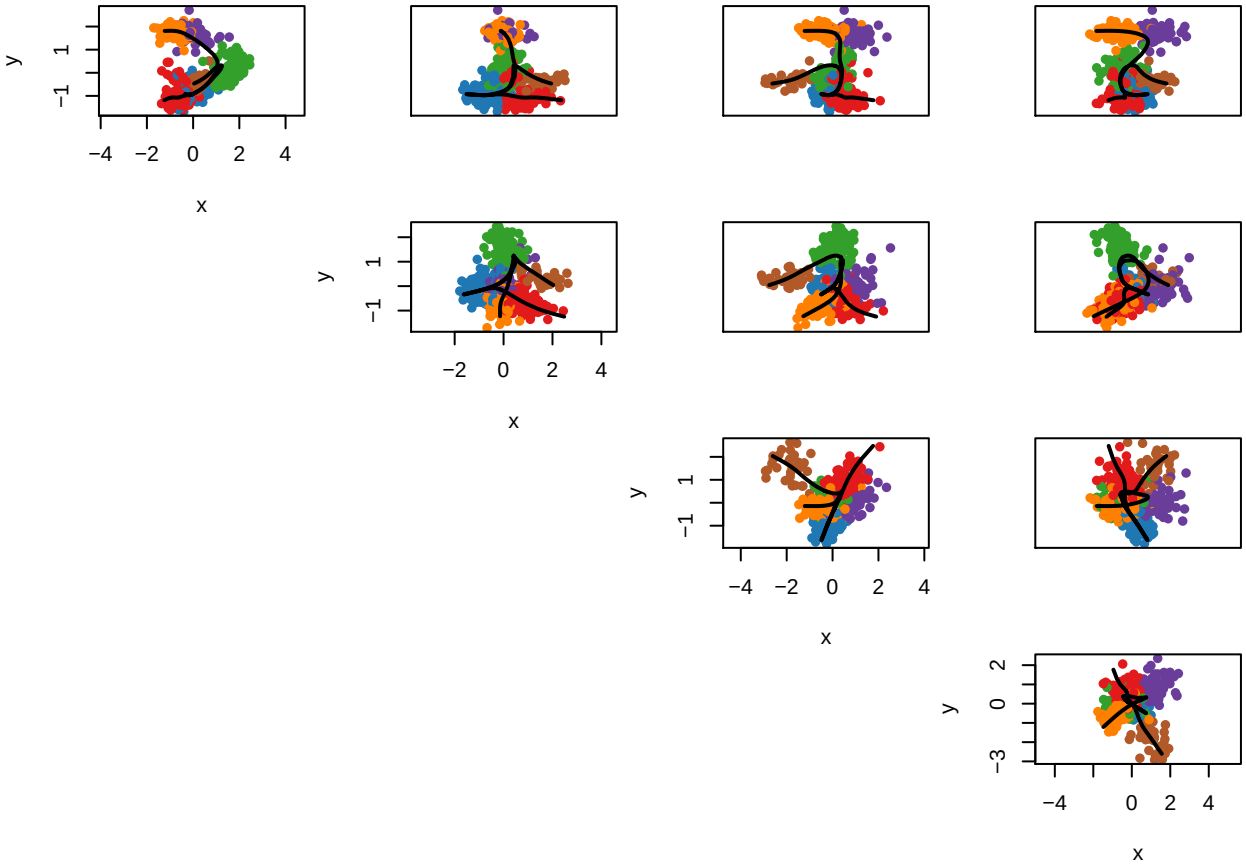

```
## [1] "K=3"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
```

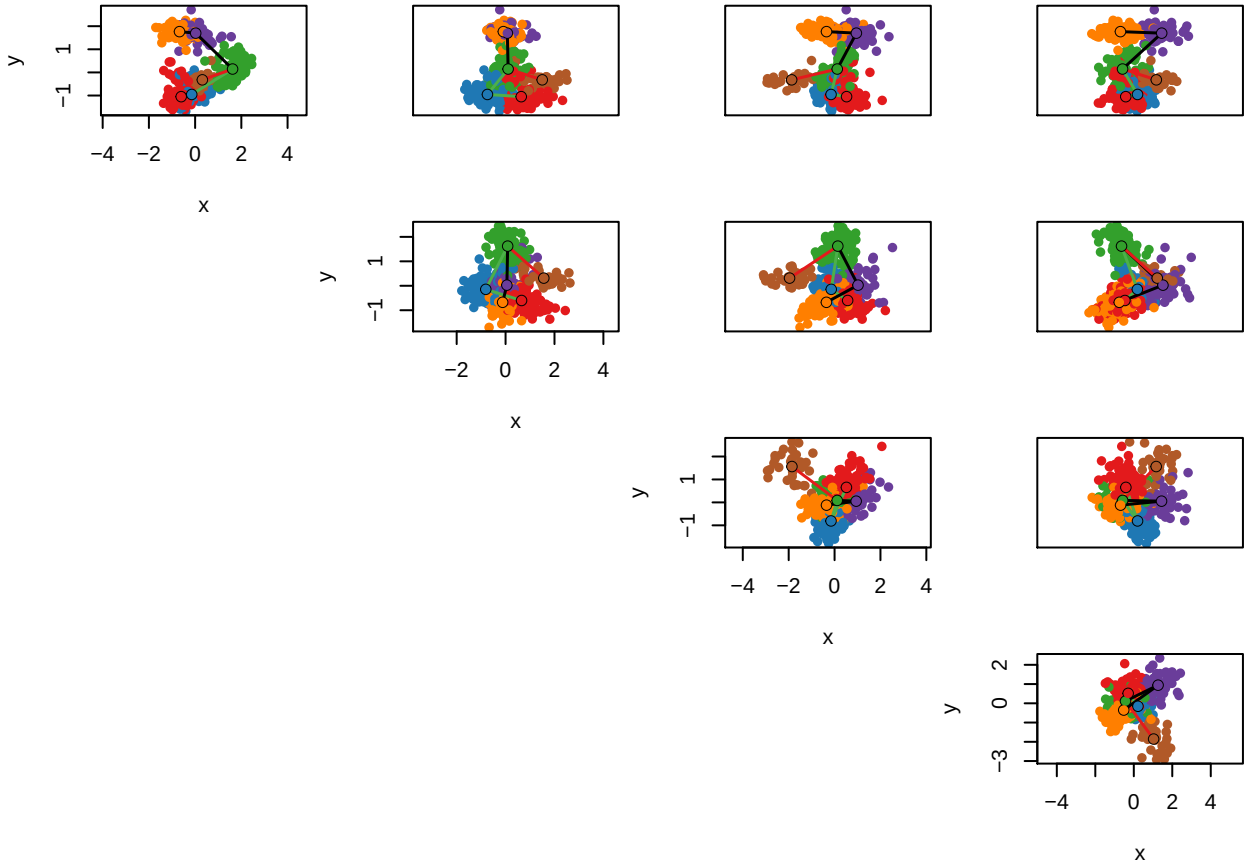
```
## NULL
## NULL
```




```
## [1] "K=4"
```

```
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```



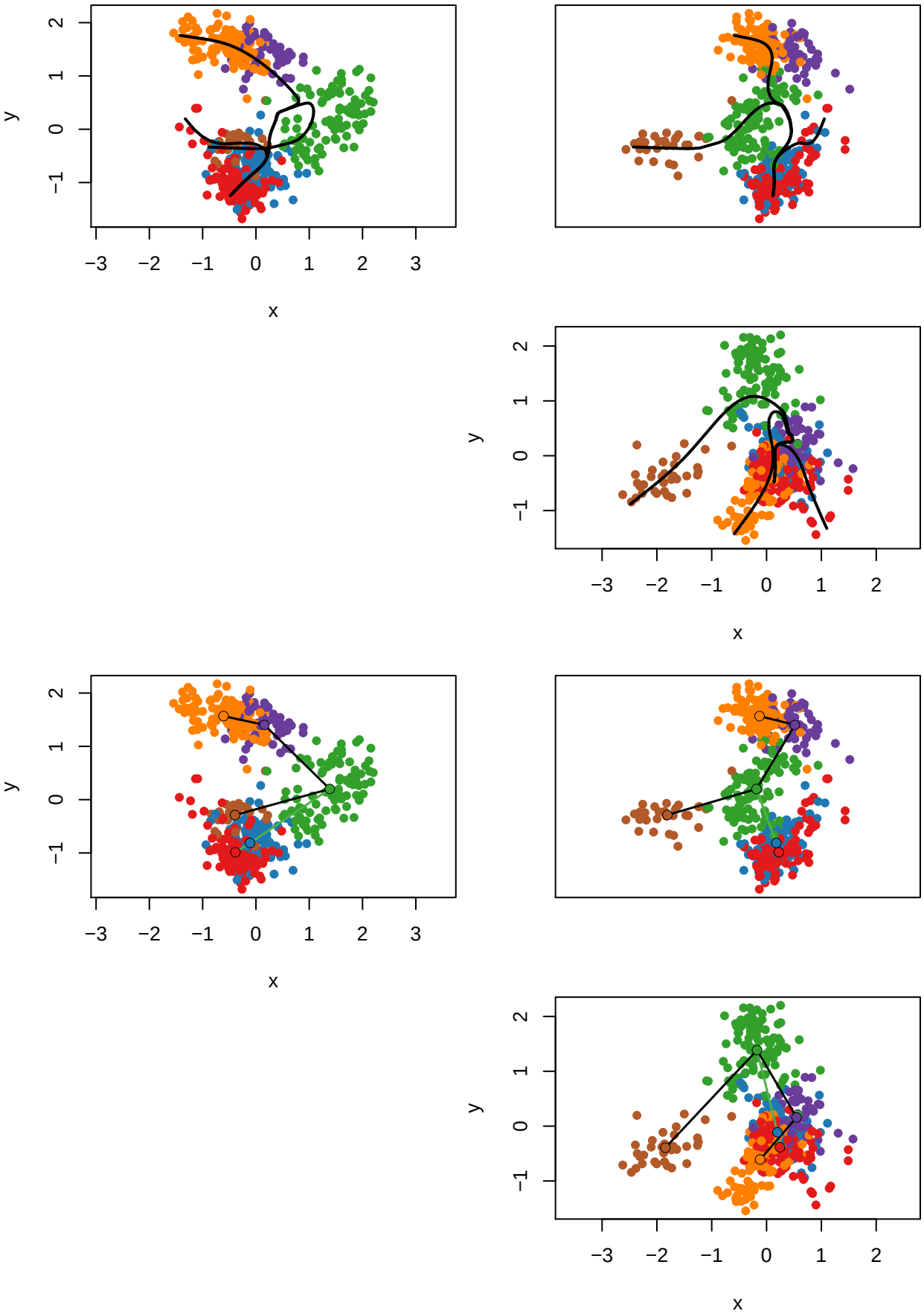



```
## [1] "K=5"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

Re-fitting zinbwave

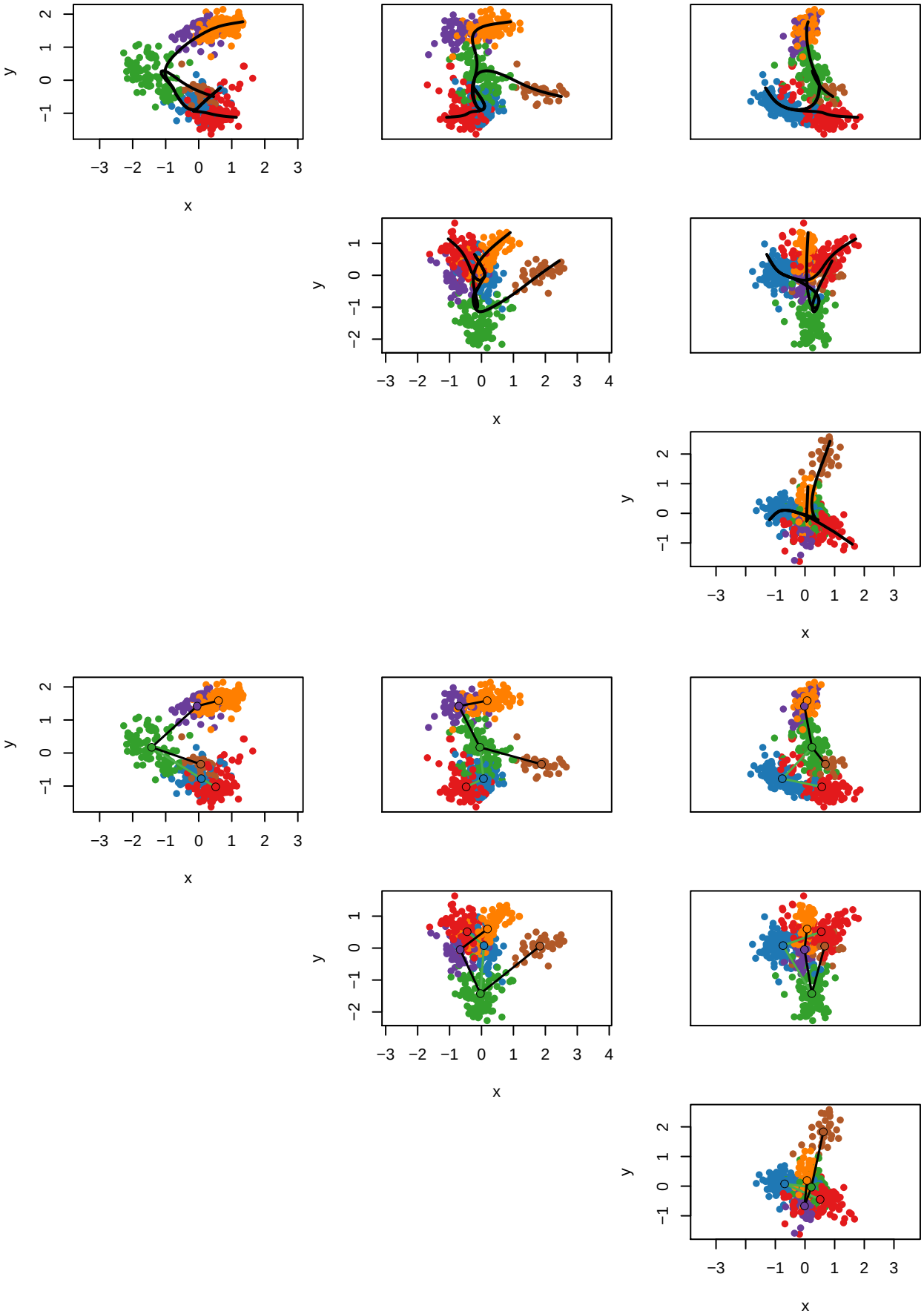
Unsupervised K = 3, 4, 5 get same results, all good.

```
for(k in Kvec) {
  X <- getW(zinbList[[k - 2]])
  mds <- cmdscale(dist(X), eig = TRUE, k = k)
  X <- mds$points

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1")
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
  print(lineages$lineage5)
}
```



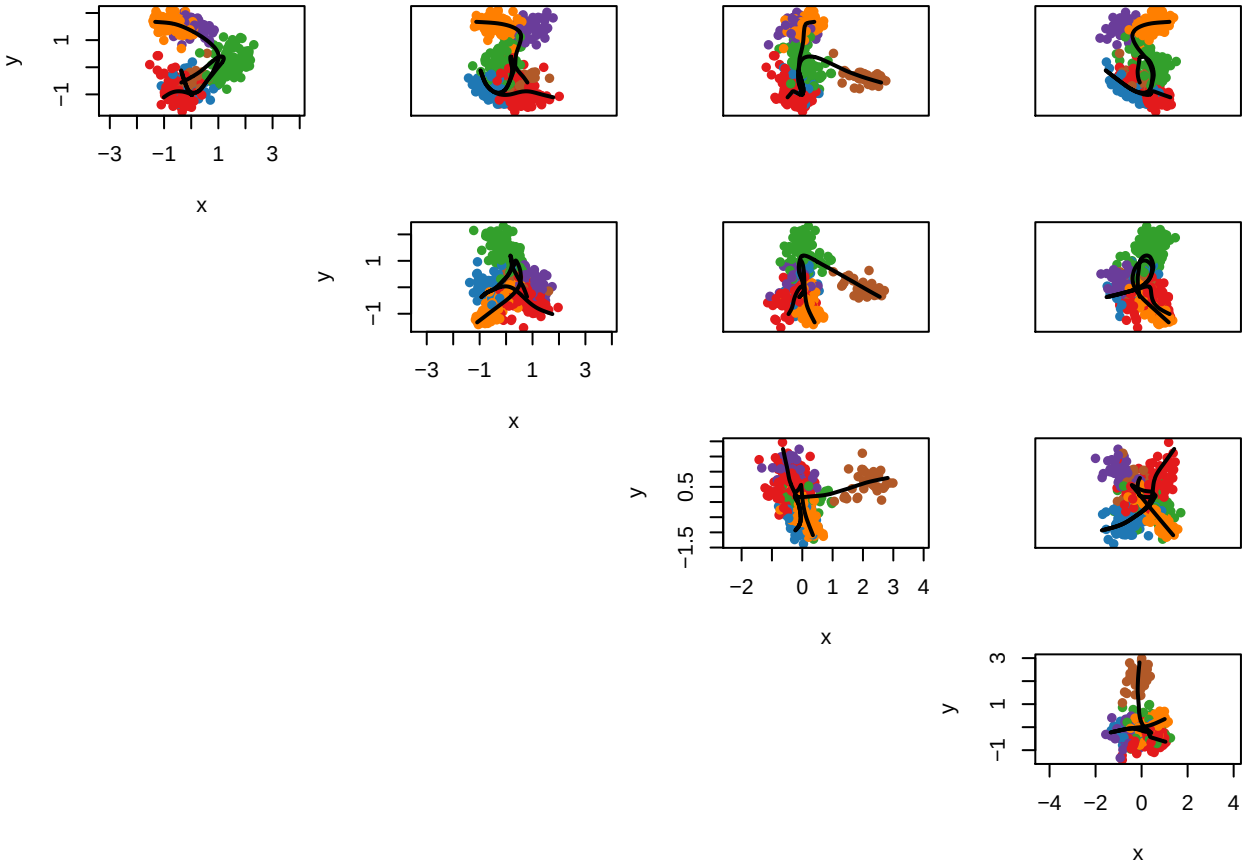

```
## [1] "K=3"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
```

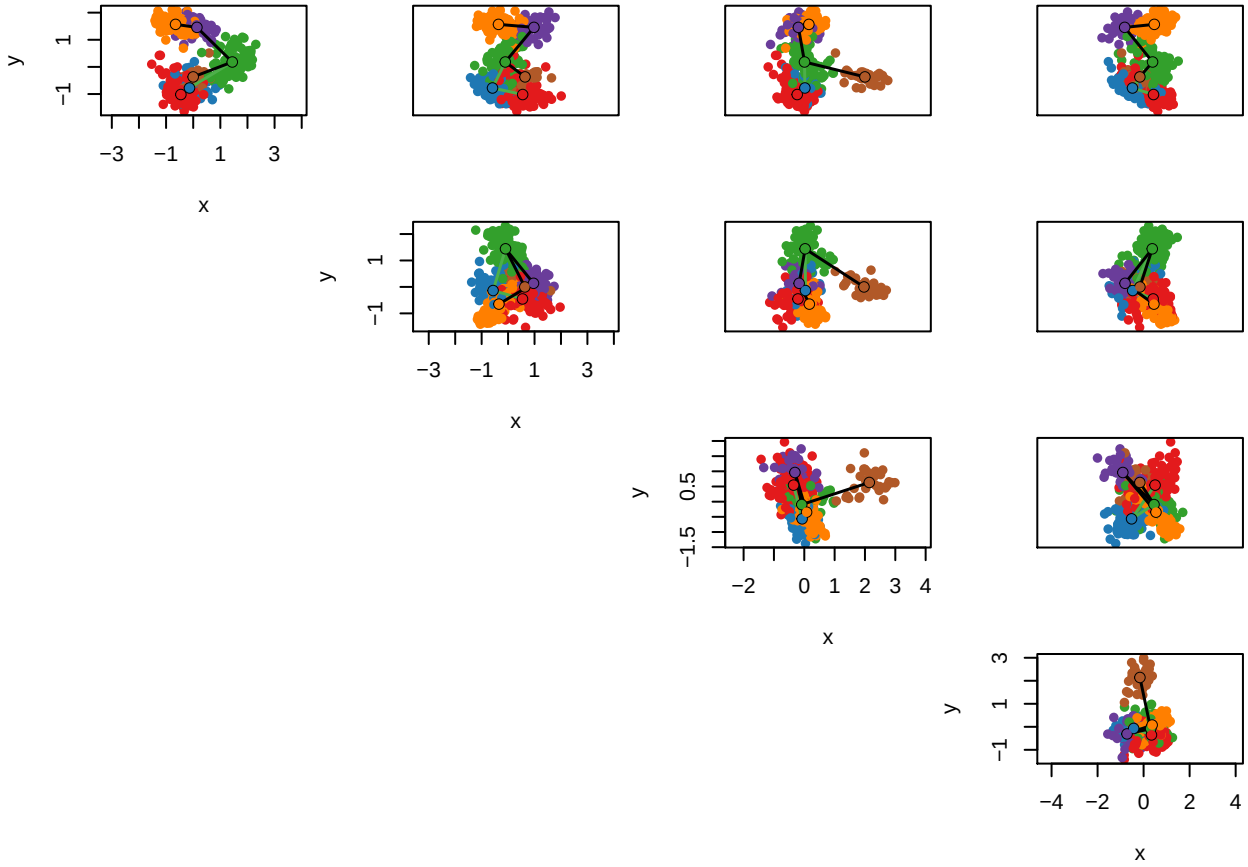
```
## NULL
## NULL
```




```
## [1] "K=4"
```

```
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

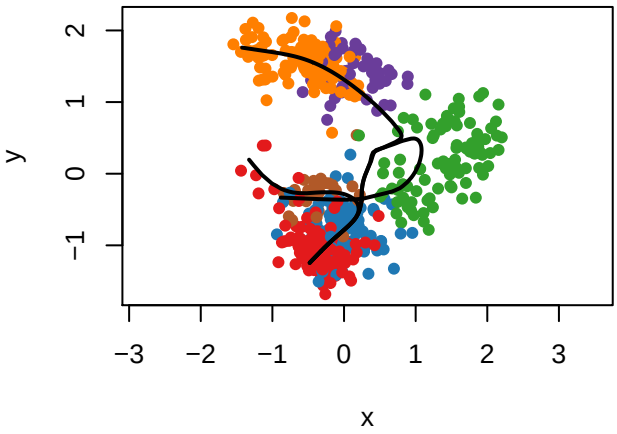


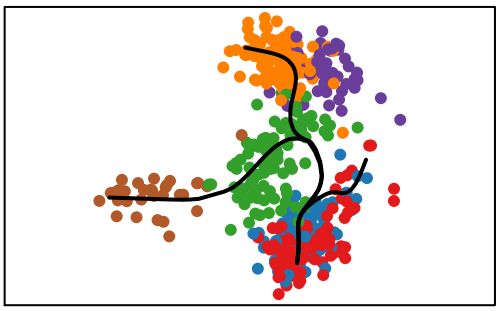


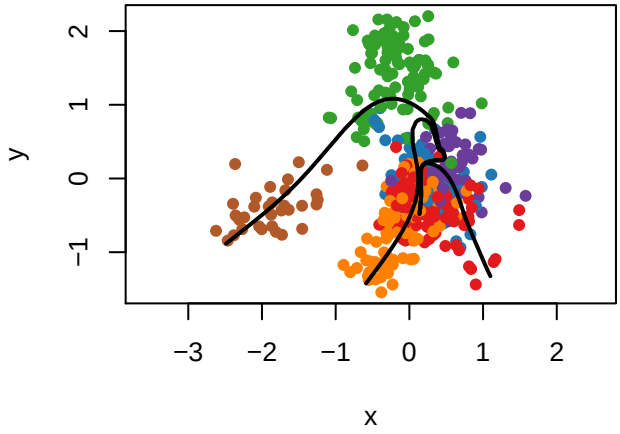

```
## [1] "K=5"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

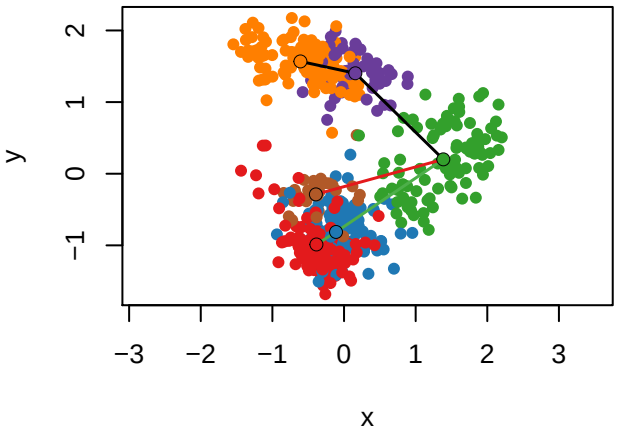
Supervised K = 3, 4, 5 get same results, all good.

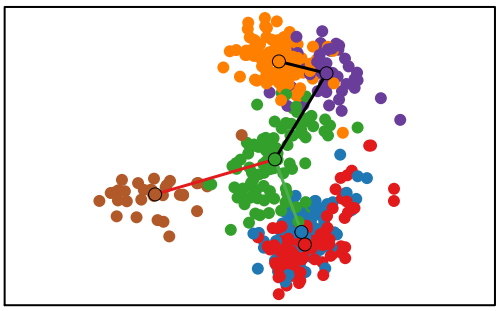
```
for(k in Kvec){
  X <- getW(zinbList[[k - 2]])
  mds <- cmdscale(dist(X), eig = TRUE, k = k)
  X <- mds$points

  lineages <- get_lineages(X, clus.labels = cl, start.clus = "c1",
                           end.clus = c("c3", "c6"))
  curves <- get_curves(X, clus.labels = cl, lineages = lineages)
  plot_curves(X, cl, curves, col.clus = pal)
  plot_tree(X, cl, lineages, col.clus = pal)

  print(paste0("K=", k))
  print(lineages$lineage1)
  print(lineages$lineage2)
  print(lineages$lineage3)
  print(lineages$lineage4)
  print(lineages$lineage5)
}
```

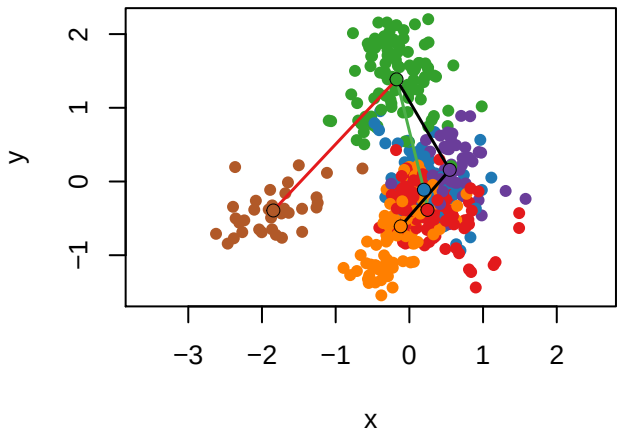






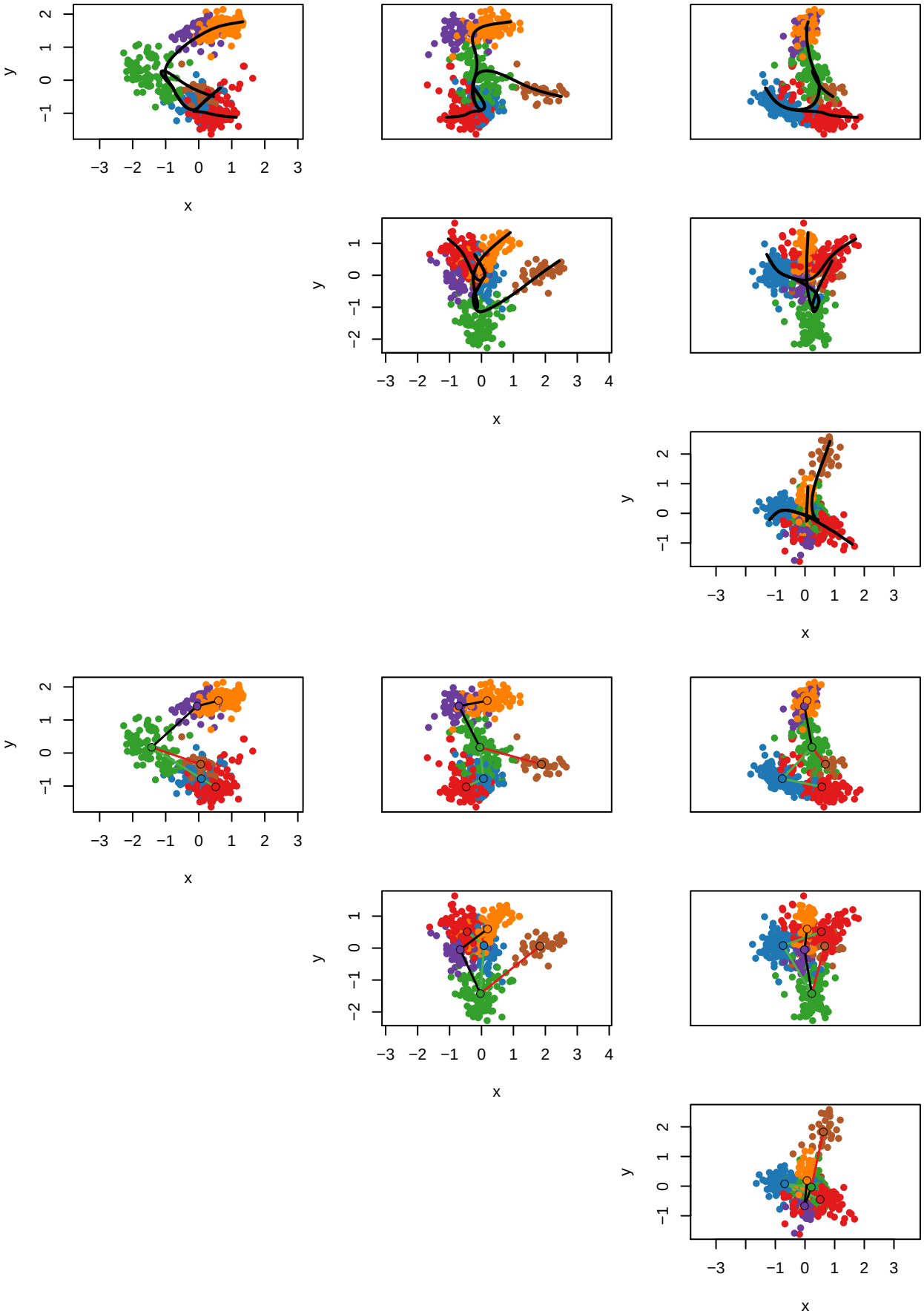






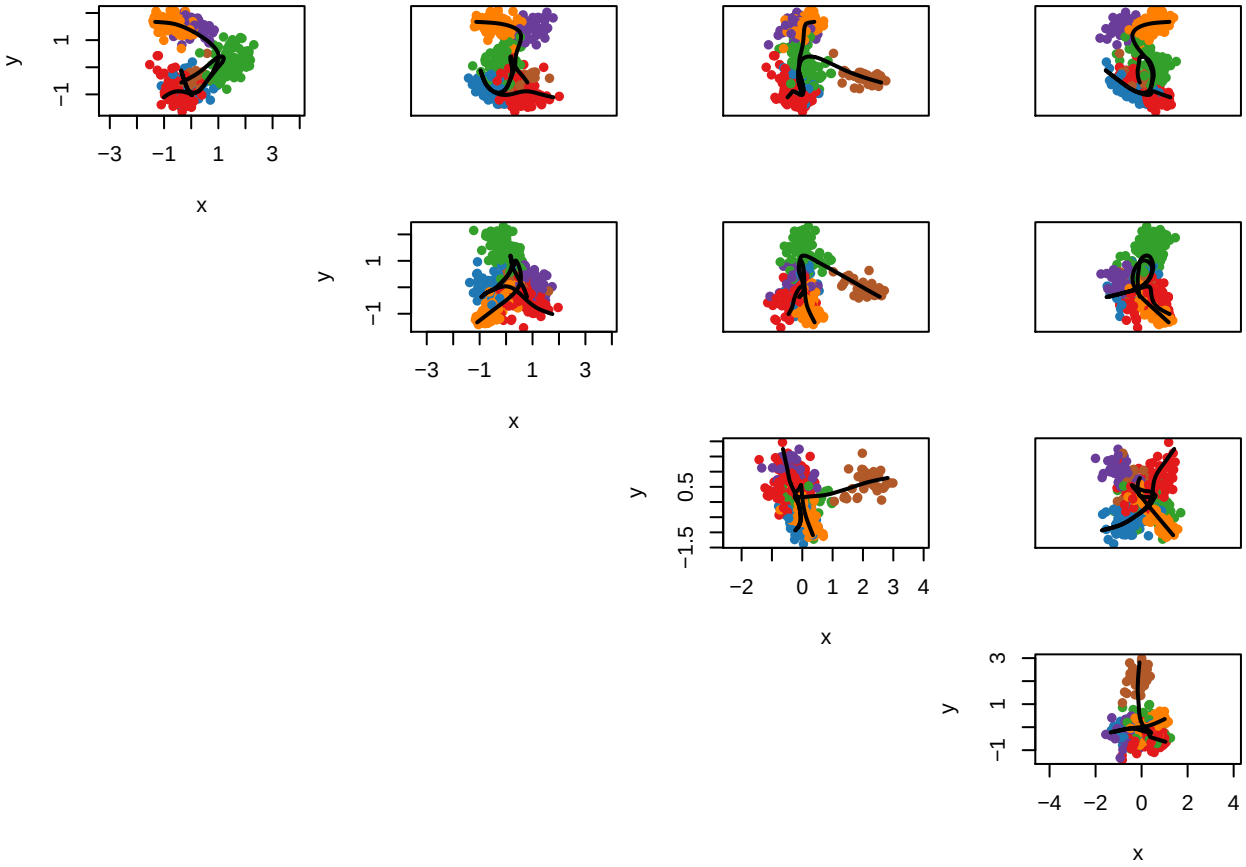

```
## [1] "K=3"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
```

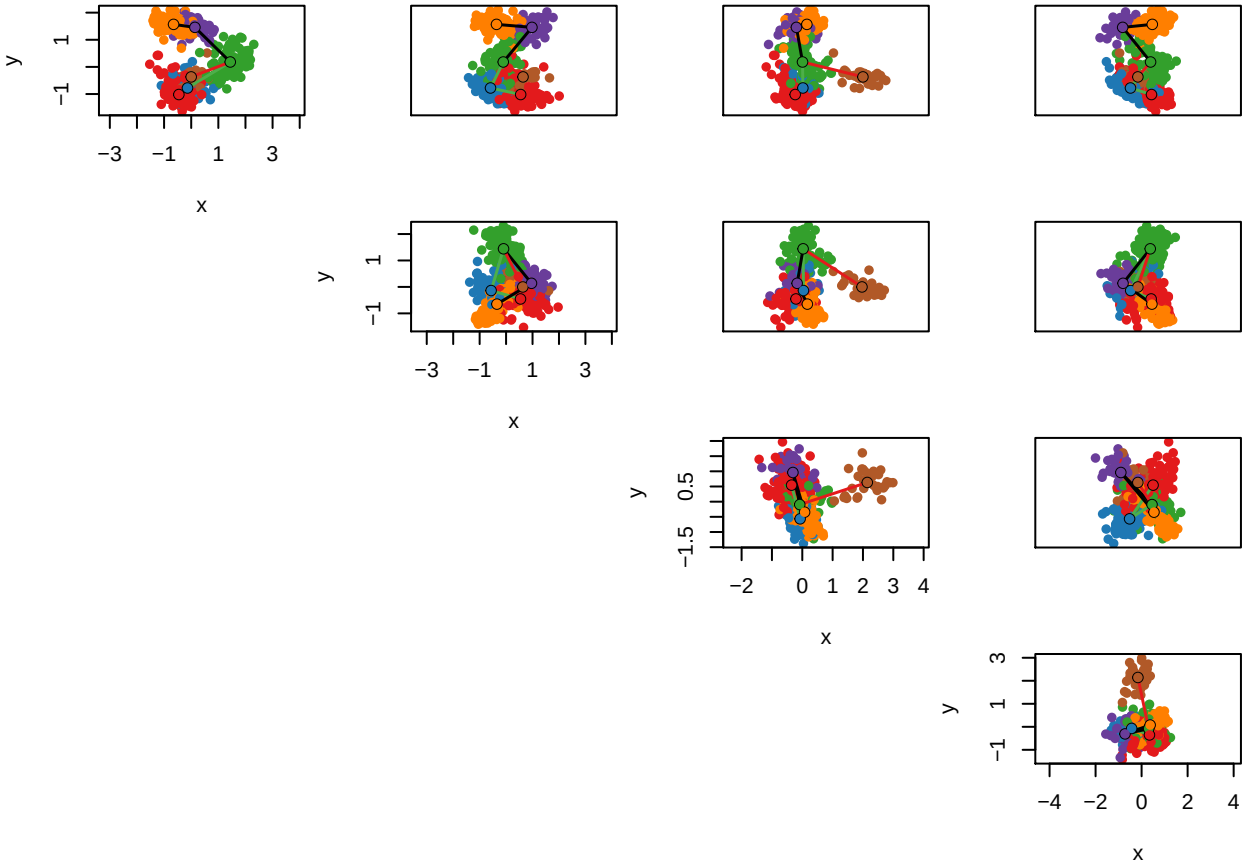
```
## NULL
## NULL
```




```
## [1] "K=4"
```

```
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```






```
## [1] "K=5"
## [1] "c1" "c2" "c5" "c4"
## [1] "c1" "c2" "c6"
## [1] "c1" "c3"
## NULL
## NULL
```

DE analysis

Here is the kind of plots we want to present

```
de <- read.csv('../data/oe_markers.txt', stringsAsFactors = F, header = F)
de <- de$V1
plotHeatmap(ceObj,
            visualizeData = assays(se)$normalizedValues[rownames(se) %in% de, ],
            clusterSamplesData = "dendrogramValue",
            whichClusters = "primary",
            main = 'Normalized values, 1000 most variable genes',
            breaks = 0.99)
```

Further developments

- DE analysis using slingshot and zinbwave
- SingleCellExperiment class

Conclusions

This workflow provides guidance to perform downstream analysis of single-cell RNA-seq datasets in R. We propose a worflow in four steps: dimensionality reduction while adjusting for gene and cell-level covariates, robust and stable clustering, pseudotime ordering, and DE analysis between the clusters. The workflow is general and flexible allowing the user to sustitute each step of the worflow with a different R tool and to input data that have been previously processed by different tools. We hope the workflow will ease the technical aspect of single-cell RNA-seq data analysis to help discovering new biological insights.

Software availability

This section will be generated by the Editorial Office before publication. Authors are asked to provide some initial information to assist the Editorial Office, as detailed below.

1. URL link to where the software can be downloaded from or used by a non-coder (AUTHOR TO PROVIDE; optional)
2. URL link to the author's version control system repository containing the source code (AUTHOR TO PROVIDE; required)
3. Link to source code as at time of publication (*F1000Research* TO GENERATE)
4. Link to archived source code as at time of publication (*F1000Research* TO GENERATE)
5. Software license (AUTHOR TO PROVIDE; required)

The three packages zinbwave, clusterExperiment, and slingshot used in the workflow are bioconductor packages and are available on github at respectively `https://github.com/drisso/zinbwave`, `https://github.com/epurdom/clusterExperiment`, and `https://github.com/kstreet13/slingshot`. The source code for this package can be found at `https://github.com/fperraudeau/singlecellworkflow` under license XXX.

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.5
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] rARPACK_0.11-0            digest_0.6.12
##  [3] RColorBrewer_1.1-2        Rtsne_0.13
##  [5] magrittr_1.5              gplots_3.0.1
##  [7] ggplot2_2.2.1             doParallel_1.0.10
##  [9] iterators_1.0.8           foreach_1.4.3
## [11] slingshot_0.0.3-5         princurve_1.1-12
## [13] zinbwave_0.99.4.2         BiocParallel_1.10.1
## [15] clusterExperiment_1.3.1   scone_1.0.0
## [17] scRNAseq_1.2.0            SummarizedExperiment_1.6.3
## [19] DelayedArray_0.2.7        matrixStats_0.52.2
## [21] Biobase_2.36.2            GenomicRanges_1.28.3
## [23] GenomeInfoDb_1.12.2       IRanges_2.10.2
## [25] S4Vectors_0.14.3          BiocGenerics_0.22.0
##
## loaded via a namespace (and not attached):
##   [1] shinydashboard_0.6.0     R.utils_2.5.0
##   [3] RSQLite_1.1-2            AnnotationDbi_1.38.1
##   [5] htmlwidgets_0.8          grid_3.4.0
##   [7] trimcluster_0.1-2        RNeXML_2.0.7
##   [9] DESeq_1.28.0             munsell_0.4.3
##  [11] codetools_0.2-15         statmod_1.4.29
##  [13] scran_1.4.4              DT_0.2
##  [15] miniUI_0.1.1             colorspace_1.3-2
##  [17] energy_1.7-0             knitr_1.16
##  [19] uuid_0.1-2               pspline_1.0-17
##  [21] robustbase_0.92-7        bayesm_3.0-2
##  [23] NMF_0.20.6               tximport_1.4.0
```

```
##  [25] GenomeInfoDbData_0.99.0  hwriter_1.3.2
##  [27] rhdf5_2.20.0             rprojroot_1.2
##  [29] EDASeq_2.10.0            diptest_0.75-7
##  [31] R6_2.2.1                 ggbeeswarm_0.5.3
##  [33] taxize_0.8.4             locfit_1.5-9.1
##  [35] flexmix_2.3-14           bitops_1.0-6
##  [37] reshape_0.8.6            assertthat_0.2.0
##  [39] scales_0.4.1             nnet_7.3-12
##  [41] beeswarm_0.2.3           gtable_0.2.0
##  [43] phylobase_0.8.4          RUVSeq_1.10.0
##  [45] bold_0.4.0               rlang_0.1.1
##  [47] genefilter_1.58.1        splines_3.4.0
##  [49] rtracklayer_1.36.3       lazyeval_0.2.0
##  [51] hexbin_1.27.1            rgl_0.98.1
##  [53] yaml_2.1.14              reshape2_1.4.2
##  [55] abind_1.4-5              GenomicFeatures_1.28.3
##  [57] backports_1.1.0          httpuv_1.3.3
##  [59] tensorA_0.36             tools_3.4.0
##  [61] bookdown_0.4             gridBase_0.4-7
##  [63] stabledist_0.7-1         dynamicTreeCut_1.63-1
##  [65] Rcpp_0.12.11             plyr_1.8.4
##  [67] progress_1.1.2           visNetwork_1.0.3
##  [69] zlibbioc_1.22.0          purrr_0.2.2.2
##  [71] RCurl_1.95-4.8           prettyunits_1.0.2
##  [73] viridis_0.4.0            zoo_1.8-0
##  [75] cluster_2.0.6            data.table_1.10.4
##  [77] RSpectra_0.12-0          mvtnorm_1.0-6
##  [79] whisker_0.3-2            gsl_1.9-10.3
##  [81] aroma.light_3.6.0        mime_0.5
##  [83] evaluate_0.10            xtable_1.8-2
##  [85] XML_3.98-1.7             mclust_5.3
##  [87] gridExtra_2.2.1          compiler_3.4.0
##  [89] biomaRt_2.32.1           scater_1.4.0
##  [91] tibble_1.3.3             KernSmooth_2.23-15
##  [93] R.oo_1.21.0              htmltools_0.3.6
##  [95] pcaPP_1.9-61             segmented_0.5-2.0
##  [97] BiocWorkflowTools_1.2.0  tidyr_0.6.3
##  [99] geneplotter_1.54.0       howmany_0.3-1
## [101] DBI_0.6-1                MASS_7.3-47
## [103] fpc_2.1-10               MAST_1.2.1
## [105] boot_1.3-19              compositions_1.40-1
## [107] ShortRead_1.34.0         Matrix_1.2-10
## [109] ade4_1.7-6               R.methodsS3_1.7.1
## [111] gdata_2.18.0             igraph_1.0.1
## [113] rncl_0.8.2               GenomicAlignments_1.12.1
## [115] registry_0.3             numDeriv_2016.8-1
## [117] locfdr_1.1-8             plotly_4.7.0
## [119] xml2_1.1.1               annotate_1.54.0
## [121] vipor_0.4.5              rngtools_1.2.4
## [123] pkgmaker_0.22            XVector_0.16.0
## [125] stringr_1.2.0            copula_0.999-16
## [127] ADGofTest_0.3            softImpute_1.4
## [129] Biostrings_2.44.1        rmarkdown_1.5
## [131] dendextend_1.5.2         edgeR_3.18.1
## [133] kernlab_0.9-25           shiny_1.0.3
## [135] Rsamtools_1.28.0         gtools_3.5.0
## [137] modeltools_0.2-21        rjson_0.2.15
## [139] nlme_3.1-131             jsonlite_1.5
## [141] viridisLite_0.2.0        limma_3.32.2
## [143] lattice_0.20-35          httr_1.2.1
## [145] DEoptimR_1.0-8           survival_2.41-3
## [147] glue_1.0.0               FNN_1.1
## [149] prabclus_2.2-6           glmnet_2.0-10
## [151] class_7.3-14             stringi_1.1.5
## [153] mixtools_1.1.0           latticeExtra_0.6-28
```

```
## [155] caTools_1.17.1           memoise_1.1.0
## [157] dplyr_0.7.0              ape_4.1
```

Author contributions

In order to give appropriate credit to each author of an article, the individual contributions of each author to the manuscript should be detailed in this section. We recommend using author initials and then stating briefly how they contributed.

Competing interests

All financial, personal, or professional competing interests for any of the authors that could be construed to unduly influence the content of the article must be disclosed and will be displayed alongside the article. If there are no relevant competing interests to declare, please add the following: 'No competing interests were disclosed'.

Grant information

Please state who funded the work discussed in this article, whether it is your employer, a grant funder etc. Please do not list funding that you have that is not relevant to this specific piece of research. For each funder, please state the funder's name, the grant number where applicable, and the individual to whom the grant was assigned. If your work was not funded by any grants, please include the line: 'The author(s) declared that no grants were involved in supporting this work.'

Acknowledgments

This section should acknowledge anyone who contributed to the research or the article but who does not qualify as an author based on the criteria provided earlier (e.g. someone or an organization that provided writing assistance). Please state how they contributed; authors should obtain permission to acknowledge from all those mentioned in the Acknowledgments section.

Fletcher, Russell B, Diya Das, Levi Gadye, Kelly N Street, Ariane Baudhuin, Allon Wagner, Michael B Cole, et al. 2017. "Deconstructing Olfactory Stem Cell Trajectories at Single-Cell Resolution." *Cell Stem Cell* 20 (6). Elsevier: 817–830.e8. doi:10.1016/j.stem.2017.04.003.

Lun, Aaron T.L., Davis J. McCarthy, and John C. Marioni. 2016. "A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor." *F1000Research* 5 (October): 2122. doi:10.12688/f1000research.9501.2.

McCarthy, Davis J., Kieran R. Campbell, Aaron T. L. Lun, and Quin F. Wills. 2017. "Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R." *Bioinformatics*, January, btw777. doi:10.1093/bioinformatics/btw777.