Single Cell Transcriptomic Analysis of 3 day old Nematostella vectensis Embryo

Carlos Roche

Table of contents


```
knitr::opts_knit$set(root.dir="/scratch/course/2023w300106/rochearcas/2part/cellranger/3d/
setwd("/scratch/course/2023w300106/rochearcas/2part/cellranger/3d/outs/filtered_feature_bc
Sys.setenv(LANG = "en")
my.sample="3d"

.libPaths(c("/lisc/user/rochearcas/R/Seurat5","/home/apps/seurat5/5.0.1-4.3.2"))
library(easypackages)
libraries("readxl","RColorBrewer",'patchwork','dplyr','viridis','ggplot2','pals','clustree
```

```
Loading required package: readxl
```

```
Loading required package: RColorBrewer
```

```
Loading required package: patchwork
```

```
Loading required package: dplyr
```

```

Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

Loading required package: viridis

Loading required package: viridisLite

Loading required package: ggplot2

Loading required package: pals


Attaching package: 'pals'

The following objects are masked from 'package:viridis':

    cividis, inferno, magma, plasma, turbo, viridis

The following objects are masked from 'package:viridisLite':

    cividis, inferno, magma, plasma, turbo, viridis

Loading required package: clustree

Loading required package: ggraph

Loading required package: ape


Attaching package: 'ape'

The following object is masked from 'package:dplyr':

    where

All packages loaded successfully
```

```
library("Seurat", lib.loc=.libPaths('/lisc/app/seurat5/5.0.1-4.3.2'))
```

```
Loading required package: SeuratObject
```

```
Loading required package: sp
```

```

Attaching package: 'sp'
```

```
The following object is masked from 'package:ggraph':

    geometry
```

```
'SeuratObject' was built with package 'Matrix' 1.6.4 but the current
version is 1.6.5; it is recomended that you reinstall 'SeuratObject' as
the ABI for 'Matrix' may have changed
```

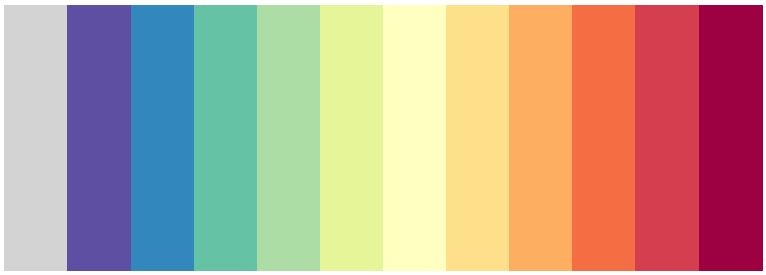
```

Attaching package: 'SeuratObject'
```

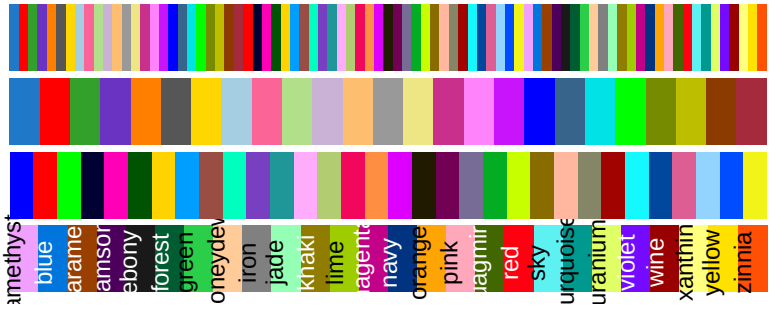
```
The following object is masked from 'package:base':

    intersect
```

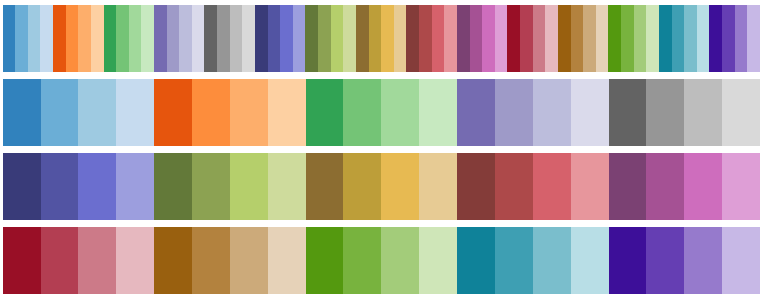
```
gene.cp = c('lightgrey', rev(brewer.pal(11 , "Spectral")))
pal.bands(gene.cp)
```



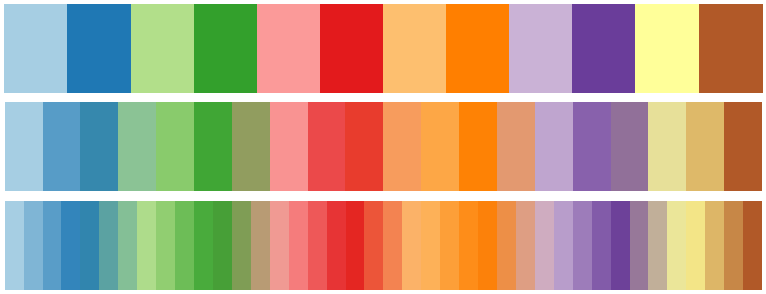
```
clust.cp.separate = unique (c(cols25(25), glasbey(32), alphabet(26)))
pal.bands(clust.cp.separate,cols25(25), glasbey(32), alphabet(26))
```



```
clust.cp.graded = unique(c(stepped3(20), stepped2(20), stepped(20)))
pal.bands(clust.cp.graded,stepped3(20), stepped2(20), stepped(20))
```



```
LibCP = brewer.paired(12)
pal.bands(LibCP,brewer.paired(20),brewer.paired(40))
```



```
load("/scratch/course/2023w300106/rochearcas/2part/results/Genes.Nv2.RData")

ld=paste('/scratch/course/2023w300106/rochearcas/2part/cellranger',my.sample,sep = '/','ou
raw.data1 <- Read10X(data.dir = as.character(ld))
if (identical(genes$geneID, rownames(raw.data1)) == F)
    #if false, something is wrong: troubleshoot before proceeding!!!
  {
    stop()
  } else {
  # set the gene model names to annotations for ease of analysis:
  rownames(raw.data1) <- genes$gene_short_name
  }
```

Data Filtering

```
keep = which(rowSums(raw.data1) >= 3)
length(keep)
```

```
[1] 16511
```

```
raw.data1 = raw.data1[keep, ]
genes <- genes[keep, ]
```

Seurat Workflow

```
data3d  <- CreateSeuratObject(counts = raw.data1, project = my.sample)
levels(data3d@meta.data$orig.ident) <- my.sample

print(FeatureScatter(data3d, feature1 = "nCount_RNA", feature2 = "nFeature_RNA") +
      geom_vline(xintercept = c(500, 40000)) +
      geom_hline(yintercept = c(300, 6000)) +
      stat_smooth(method = lm) +
      scale_x_log10() +
      scale_y_log10()
    )
```

```
`geom_smooth()` using formula = 'y ~ x'
```

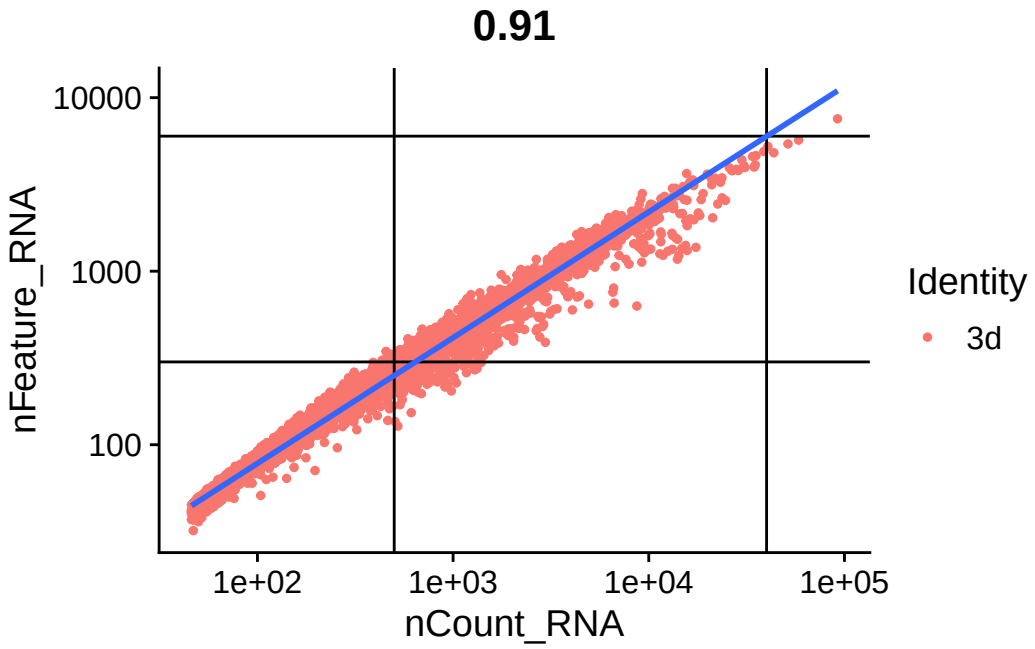


Figure 1. Plotting of total number of unique genes per cell against the total number of molecules per cell. The right side of the plot, where more molecules are detected and dots diverge from the diagonal, may correspond with multiplets.

```
data3d <- subset(x = data3d, subset = nFeature_RNA > 100 & nCount_RNA > 500)
vl = VlnPlot(data3d, features = c('nFeature_RNA', 'nCount_RNA'),log=T)
```

```
Warning: Default search for "data" layer in "RNA" assay yielded no results;
utilizing "counts" layer instead.
```

```
p1 = data3d@meta.data %>%
    ggplot(aes(x = nFeature_RNA)) +
    geom_density() +
    # scale_x_log10() +
    theme_classic() +
    ylab("Cell density") +
    geom_vline(xintercept = c(300, 4000)) #can adjust this

p2 = data3d@meta.data %>%
    ggplot(aes(x = nCount_RNA)) +
    geom_density() +
    # scale_x_log10() +
    theme_classic() +
    ylab("Cell density") +
    geom_vline(xintercept = c(750, 25000))
print(vl + p1 + p2)
```

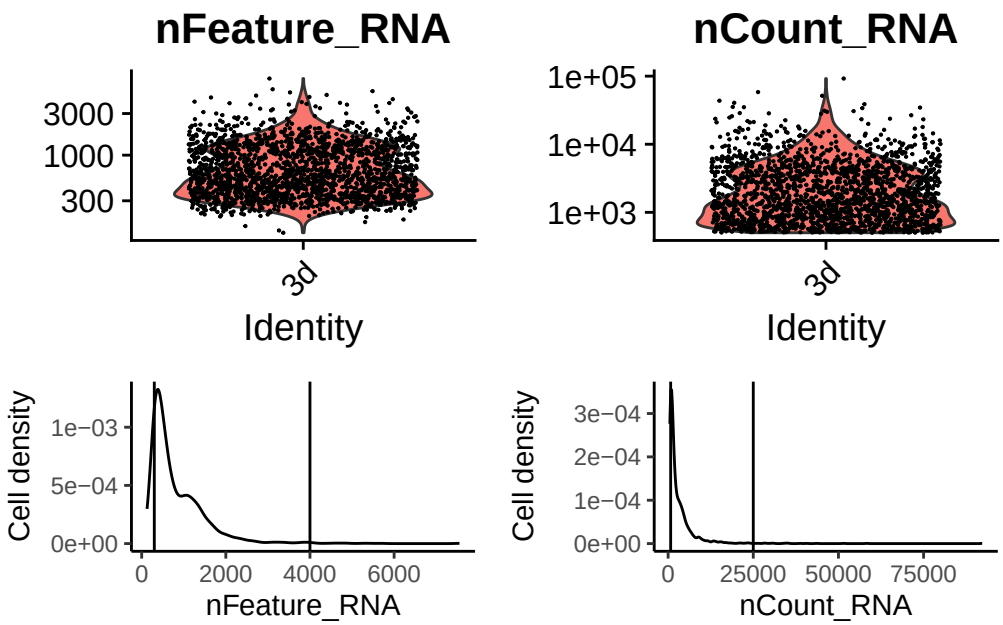


Figure 2. Violin plots and cell density plots for total genes detected per cell and total number of molecules detected per cell.

```
data3d <- subset(x = data3d, subset = nFeature_RNA > 300 & nCount_RNA < 25000)

data3d <- RenameCells(data3d, add.cell.id = my.sample)
data3d <- NormalizeData(data3d, scale.factor = 10000)
```

```
Normalizing layer: counts
```

```
data3d <- FindVariableFeatures(data3d, nfeatures = 2000)
```

```
Finding variable features for layer counts
```

```
VariableFeaturePlot(data3d)
```

```
Warning: Transformation introduced infinite values in continuous x-axis
```

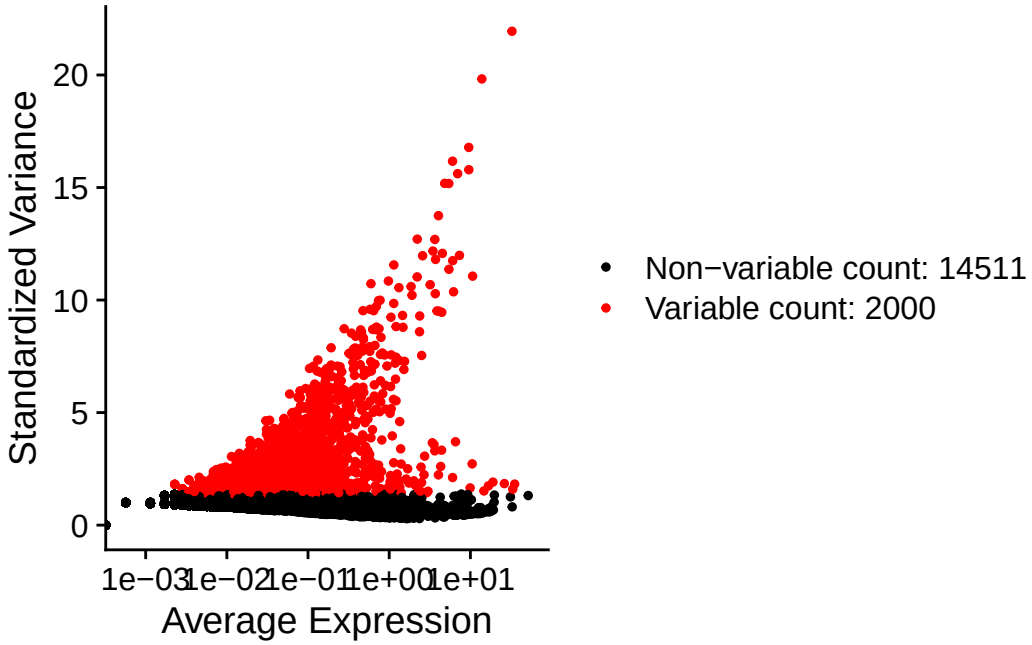


Figure 3. Standardised variance calculated for each gene across cells. A minimum value of 300 genes and a maximum value of 25000 for unique molecules per cell were set as filtering for further analysis based on visual observation of outliers on the violin plots. The 2000 most variable genes in expression across cells were selected for plotting.

```
data3d <- ScaleData(data3d)
```

```
Centering and scaling data matrix
```

```
data3d <- RunPCA(data3d, pcs.compute = 50)
```

```
PC_ 1
Positive:  ACTC-like-6, ACTP-like-1, NV2.9154, ADF9-like-1, Actin6, NV2.9829, SCRY2-like-1, I
       MMP17-like-2, STOM-like-1, NV2.6264, NV2.9832, ELEL-like-2, NV2.17435, Wnt4, HSP71-lik
       NV2.13117, NV2.13863, MMPi, NV2.10891, NV2.22443, HSPB1-like-1, TRAF3-like-1, HMCN2-li
Negative:  NV2.17003, NV2.10343, NV2.12690, NFH-like-4, GGT1-like-26, DKK3-like-2, NV2.9649,
       NV2.24598, NV2.12539, NV2.18954, ISP2-like-1, NV2.11821, NV2.9741, EVA1C-like-3, DKK3-
       NV2.16646, NV2.16524, NV2.9738, ECE2-like-6, COHA1-like-2, Ncol1, NV2.25695, NV2.13045
PC_ 2
Positive:  MCTP1-like-1, ANO8-like-1, NV2.24075, NV2.160, TRAF4-like-19, FOS-like-1, IHH-like
```

```
       DIRC2-like-17, BND7A-like-1, BIR7B-like-1, NV2.14631, NV2.13307, GSHB-like-1, NUPR1-li
       NV2.8832, T214B-like-1, NV2.25228, NV2.14410, MBOA2-like-1, KCNK9-like-8, TBA-like-2,
Negative:  GAPR1-like-57, FKBP2-like-1, NEP4, Ncol4, NV2.17017, GGHA-like-6, FBP1-like-10, NV
       TYRO-like-1, MATN2-like-9, NV2.11825, FBP1-like-45, NV2.11824, NV2.3555, NV2.9748, NOT
       GAPR1-like-53, NV2.23281, NV2.17392, EVA1C-like-3, NV2.17391, NV2.12534, GAPR1-like-17
PC_ 3
Positive:  MSRE-like-3, PAX2A-like-1, NV2.23423, TXD12-like-2, ARSB-like-1, NV2.3555, NV2.173
       VGLU1-like-7, NV2.10282, VGLU3-like-1, COT2-like-1, GILT-like-1, P4HTM-like-1, KCNK9-l
       NV2.6073, TMEDA-like-1, FOLH1-like-5, TRIM2-like-6, GGT1-like-8, G3ST1-like-3, FUT10-l
Negative:  NV2.12553, NV2.11394, NV2.9507, NV2.5190, NV2.12691, JUN-like-11, VMP6, NV2.17398
       ATF3-like-1, NV2.17397, FOS-like-4, VGLU1-like-5, Y5713-like-1, PDIA4-like-2, GAPR1-li
       ELEL-like-1, NV2.17401, HMU-like-41, NV2.12708, NV2.2533, NV2.17400, CBPD-like-2, CO6-
PC_ 4
Positive:  NV2.160, IHH-like-6, Actin6, ANO8-like-1, MBOA2-like-1, NV2.9829, NV2.9154, CALM3-
       BND7A-like-1, BIR7B-like-1, TRAF4-like-19, SEGN-like-4, T214B-like-1, MCTP1-like-1, TC
       TCAIM-like-1, NUPR1-like-1, NV2.18050, ACTC-like-6, NV2.25228, MSRE-like-3, NV2.14410
Negative:  CD151-like-2, TET5C-like-1, BASS4-like-3, R3HD2-like-1, NV2.4247, NV2.22820, NV2.1
       CRAM-like-6, NV2.3425, SEGN-like-1, Mucin, SPEE-like-2, ALK-like-1, NEC1-like-2, FKB14
       NV2.25322, NV2.7431, B4GT1-like-1, CBX2-like-1, NV2.10280, NV2.4059, DOX2-like-2, NV2
PC_ 5
Positive:  CBPA2-like-3, NV2.25475, CBPA4-like-1, BMP1-like-2, KLKB1-like-10, NV2.16893, CTRB
       NV2.14102, MMP19-like-1, SAP3-like-13, HCE1-like-1, NV2.1305, MLRP2-like-4, NV2.23301
       NV2.1150, CTRB-like-7, TYR3-like-2, NAS13-like-16, MLRP1-like-31, NV2.16825, NV2.8640
Negative:  AHNK-like-9, NV2.3425, SPEE-like-2, NV2.22820, CRAM-like-6, NV2.21610, Mucin, ALK-
       NV2.3667, B4GT1-like-1, CDCP2-like-10, ELEL-like-3, NV2.7431, NV2.16639, OGFR-like-6,
       NPFF2-like-78, DOX2-like-2, GUNA-like-1, creb-like2, LAF3-like-3, NV2.19510, THIO-like
```

```
PCAPlot(data3d)
```

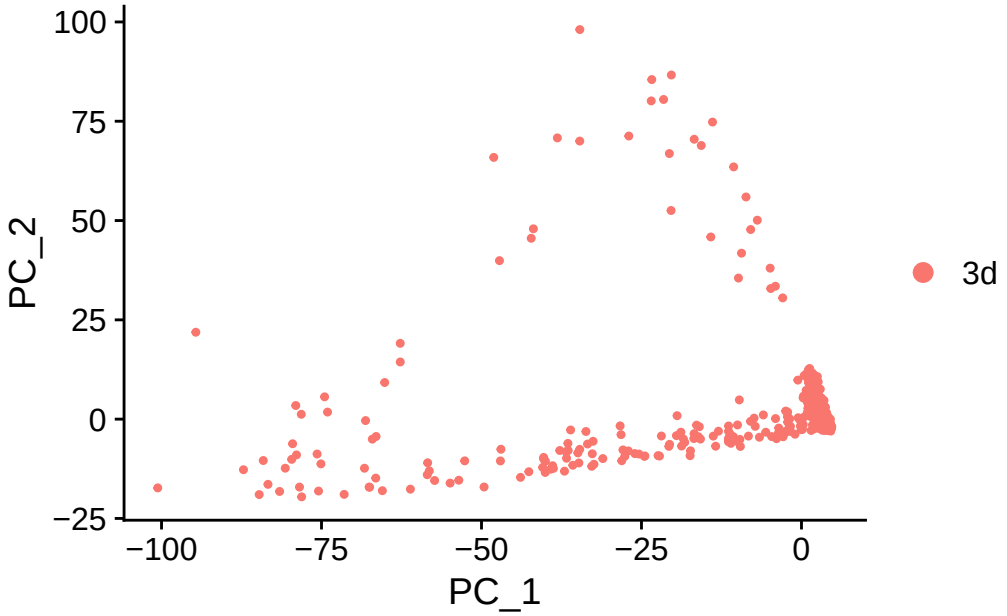


Figure 4. 2D representation of the first 2 principal component loadings based on a 50 dimensions principal component analysis on the single-cell RNA-seq data. These two PC explain the largest percentage of variance of the original sample.

```
PCHeatmap(data3d, dims = 1:3, nfeatures = 10)
```

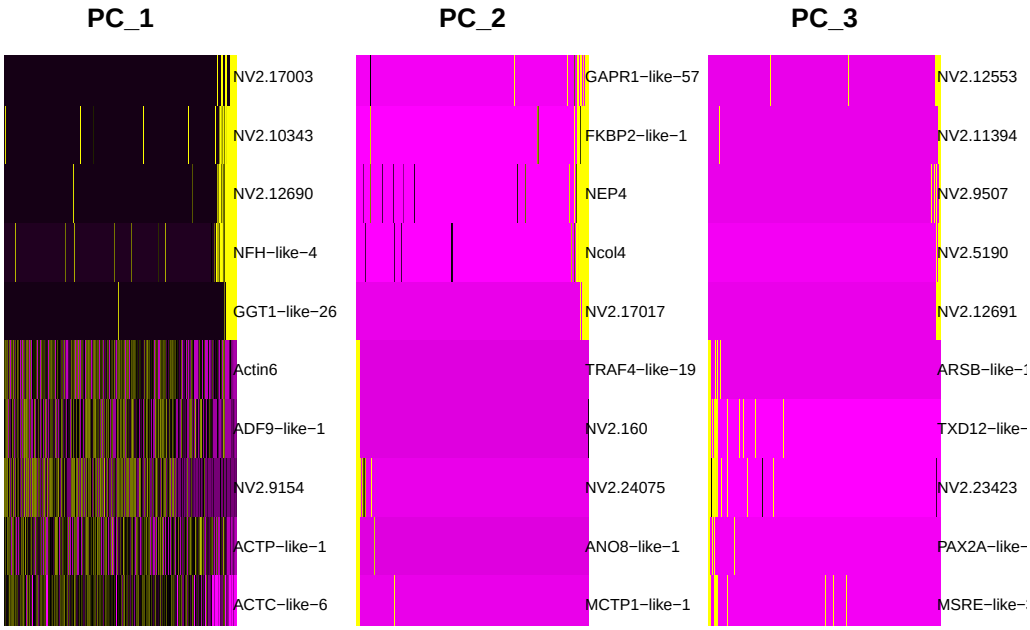


Figure 5. Loadings of principal components for each cell and gene. For each of the three most variance-explaining PCs, the 10 most differentially expressed genes are shown as loadings on the PC.

```
print(ElbowPlot(object = data3d, ndims = 50) +
          geom_hline(yintercept = 2))
```

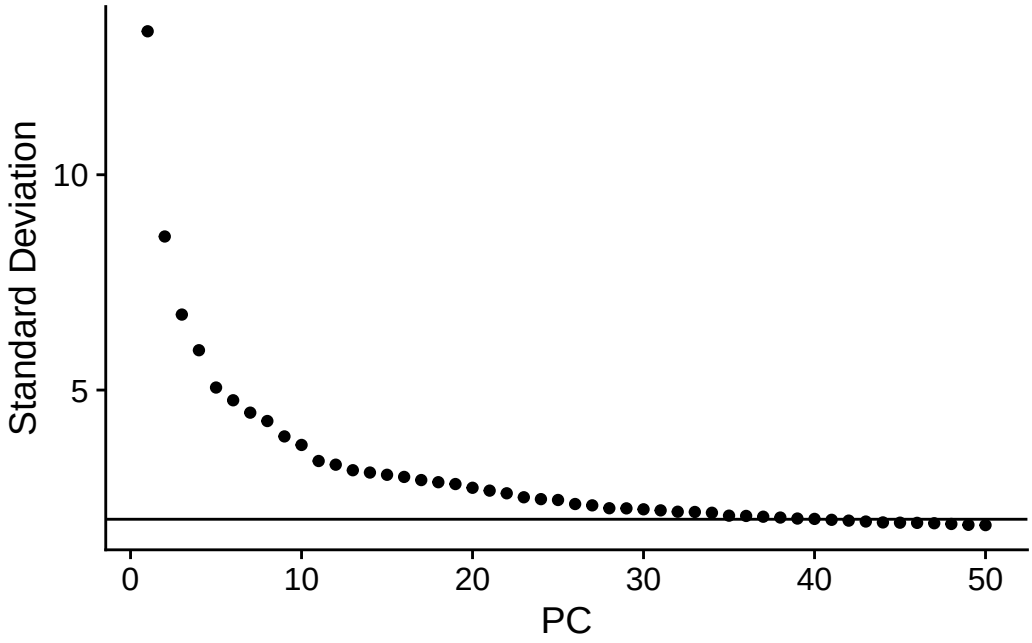


Figure 6. Elbow plot showing the explained standard deviation (SD) by PC in a 50-dimensional PCA. PCs are ordered in descending order of explained variance / SD. In order to aid in selecting informative dimensions, a threshold of 2 in explained SD was arbitrarily chosen.

```
d = c(1:10)
d = as.integer(which(data3d@reductions$pca@stdev > 2))
pct <- data3d[["pca"]]@stdev / sum(data3d[["pca"]]@stdev) * 100
```

After running jackstraw test, I decided on 11 informative dimensions in the PCA based on a standard deviation greater than 2 for a dimension to be informative.

Cell Plots - tSNE and UMAP

```
### UMAP ----
  data3d <- RunUMAP(
    data3d,
    dims = d,
    reduction = 'pca',
    reduction.name = 'umap',
    reduction.key = 'umap',
    #this is default; can change name to save different maps
    #the following parameters control your output.
```

```
    seed.use = 10,
    #random starting point; without your maps will differ every time!
    n.neighbors = 20L,
    #how many similar cells do you expect
    spread = 0.3,
    #how big is the axis space: 0.2 for lineages; 1 for separation
    min.dist = 0.3,
    #how close to plot each cell higher=spread
    local.connectivity = 30
  )#overall how connected is the graph
```

```
Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate t
To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlatio
This message will be shown once per session

19:41:06 UMAP embedding parameters a = 11.97 b = 1.929

19:41:06 Read 1753 rows and found 40 numeric columns

19:41:06 Using Annoy for neighbor search, n_neighbors = 20

19:41:06 Building Annoy index with metric = cosine, n_trees = 50

0%   10   20   30   40   50   60   70   80   90   100%

[----|----|----|----|----|----|----|----|----|----|

**************************************************|
19:41:07 Writing NN index file to temp file /tmp/RtmpgeuZBM/file145582535d6946
19:41:07 Searching Annoy index using 1 thread, search_k = 2000
19:41:07 Annoy recall = 100%
19:41:07 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
19:41:07 1753 smooth knn distance failures
19:41:08 Initializing from normalized Laplacian + noise (using RSpectra)
19:41:08 Commencing optimization for 500 epochs, with 49054 positive edges
19:41:15 Optimization finished

Warning: Keys should be one or more alphanumeric characters followed by an
underscore, setting key from umap to umap_
```

```
### tSNE ----
data3d <- RunTSNE(
  data3d,
  seed.use = 42,
  #necessary for reproducibility; try turning it off!
  perplexity = 100,
  #default 30; higher maintains greater local
  dims.use = d
)

#check that output is not based only on information content:
print(
  FeaturePlot(
    data3d,
    'nFeature_RNA',
    reduction = 'umap',
    cols = c('lightgrey', 'darkred')
  ) + NoAxes() + NoLegend() +
    labs(title = 'nFeatures', subtitle = 'umap') &
    FeaturePlot(
     data3d,
      'nFeature_RNA',
      reduction = 'tsne',
      cols = c('lightgrey', 'darkblue')
    ) + NoAxes() + NoLegend() +
    labs(title = 'nFeatures', subtitle = 'tsne')
)
```

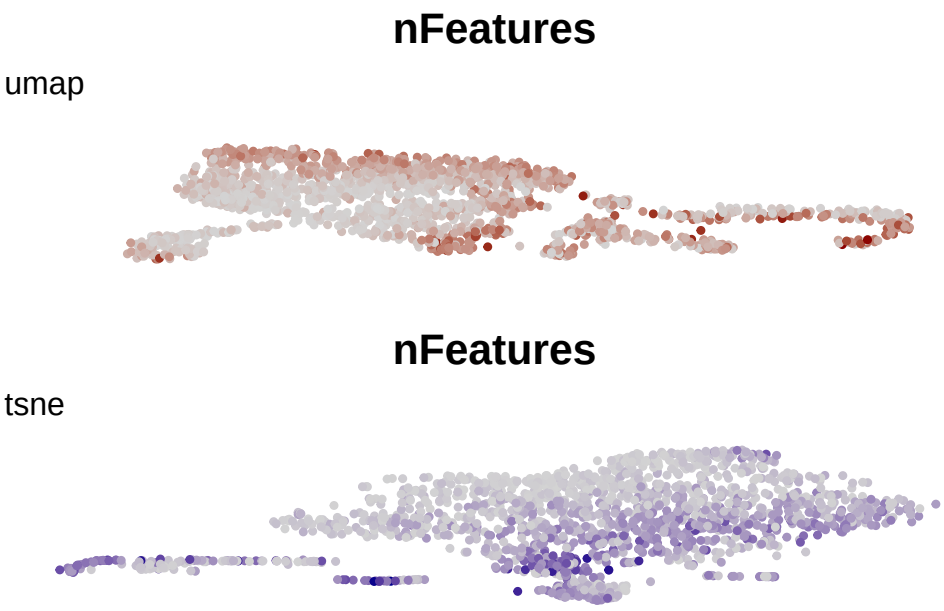


Figure 7. UMAP and tSNE cell plots for 3-day-old Nematostella vectensis embryos. Parameters perplexity for tSNE, number of neighbours, spread and minimum distance for UMAP were optimised for a clear viewing of the dataset, not compromising the global image but also to see cell clusters. These parameters were optimised only for the working dataset. Colouring in both plots represents the total number of genes detected for each cell.

Cell Clustering

A higher k.param results in a larger distance to find a neighbour to categorise as the same cluster. In the end, k.param was left as 10 as it led to the clearest clustering.

```
data3d <- FindNeighbors(
    object = data3d,
    reduction = "pca",
    dims = d,
    #this parameter is what UMAP uses:
    annoy.metric = 'cosine',
    #default = 'euclidean'
    # see https://www.nature.com/articles/s41592-019-0372-4
    k.param = 10
  )
```

```
Computing nearest neighbor graph
```

```
Computing SNN
```

```
check.resolution = T # a short script to test and plot different resolutions:
  if (check.resolution)
  {
    #here you can calculate different clustering resolutions:

    res = c(0.1, 0.5, 1, 1.5, 2) #list of resolutions 0.1 to 2 in 0.5 increments
    p = NULL
    for (i in 1:length(res))
    {
      data3d <-
        FindClusters(
          object = data3d,
          resolution = res[i],
          random.seed = 0
        )
      #save all the plots:
      p[[i]] = DimPlot(
        data3d,
        label = T,
        label.size = 5,
        repel = T,
        cols = clust.cp.separate
      ) + NoLegend() + NoAxes() +
        labs(title = c(res[i], 'Clusters | ID'))
    }

    #can use clustree to image clustering stability:
    x = clustree::clustree(data3d, prefix = "RNA_snn_res.")
    print(x)
  }
```

```
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9578
Number of communities: 7
```

```
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.8904
Number of communities: 14
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.8443
Number of communities: 20
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.8057
Number of communities: 20
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.7695
Number of communities: 24
Elapsed time: 0 seconds
```

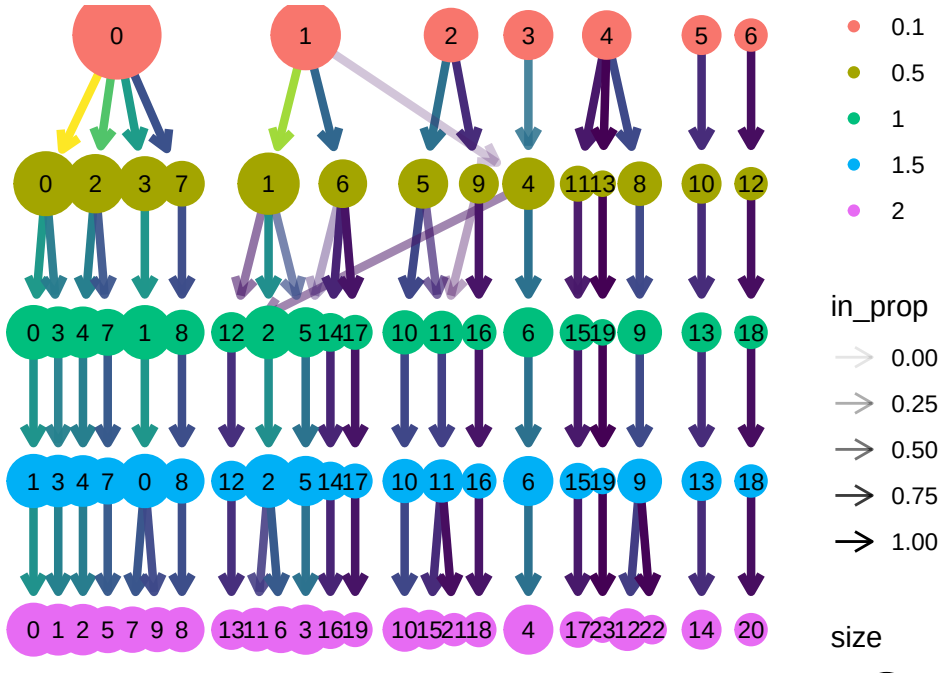


Figure 8. Clustree plot on clustering resolutions of 0.1, 0.5, 1, 1.5 and 2. Resolutions greater than 1 were considered as overclustering of the sample.

```
#choose your desired resolution:
    p[[1]] + p[[2]] + p[[3]] + p[[4]]
```

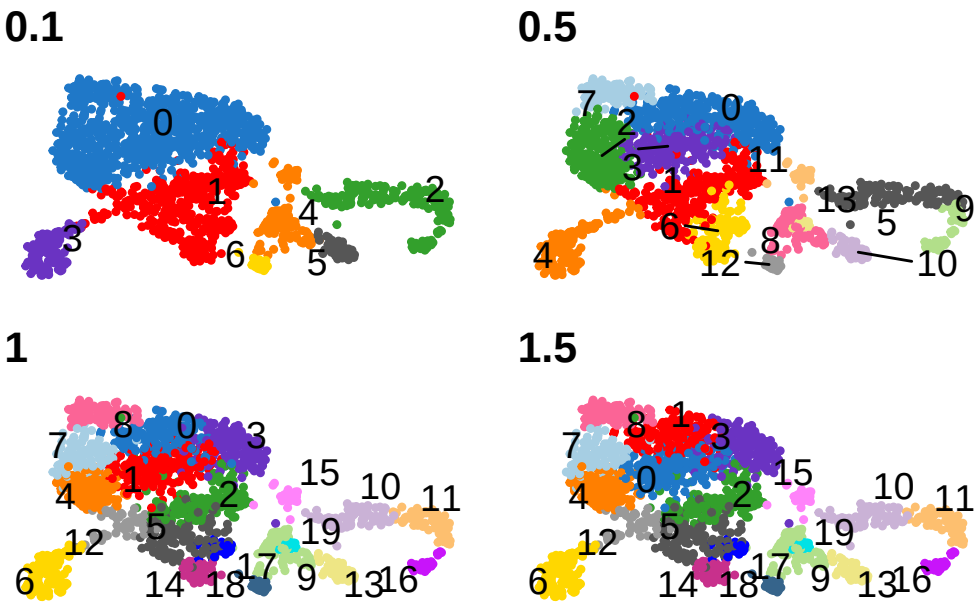


Figure 9. Cluster cell plots on resolutions 0.1, 0.5, 1 and 1.5. Clustering between 0.1 and 0.5 was observed to be more realistic, but a more accurate repetition of resolution optimisation was carried within this range.

```
res = c(0.1, 0.2, 0.3, 0.4, 0.5) #list of resolutions 0.1 to 0.5 in 0.1 increments
    p = NULL
    for (i in 1:length(res))
    {
      data3d <-
        FindClusters(
          object = data3d,
          resolution = res[i],
          random.seed = 0
        )
      #save all the plots:
      p[[i]] = DimPlot(
        data3d,
        label = T,
        label.size = 5,
        repel = T,
        cols = clust.cp.separate
      ) + NoLegend() + NoAxes() +
```

```
        labs(title = c(res[i], 'Clusters | ID'))
    }
```

```
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9578
Number of communities: 7
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9376
Number of communities: 10
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9206
Number of communities: 11
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9053
Number of communities: 13
Elapsed time: 0 seconds
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
```

```
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.8904
Number of communities: 14
Elapsed time: 0 seconds
```

```
#can use clustree to image clustering stability:
x = clustree::clustree(data3d, prefix = "RNA_snn_res.")
print(x)
```

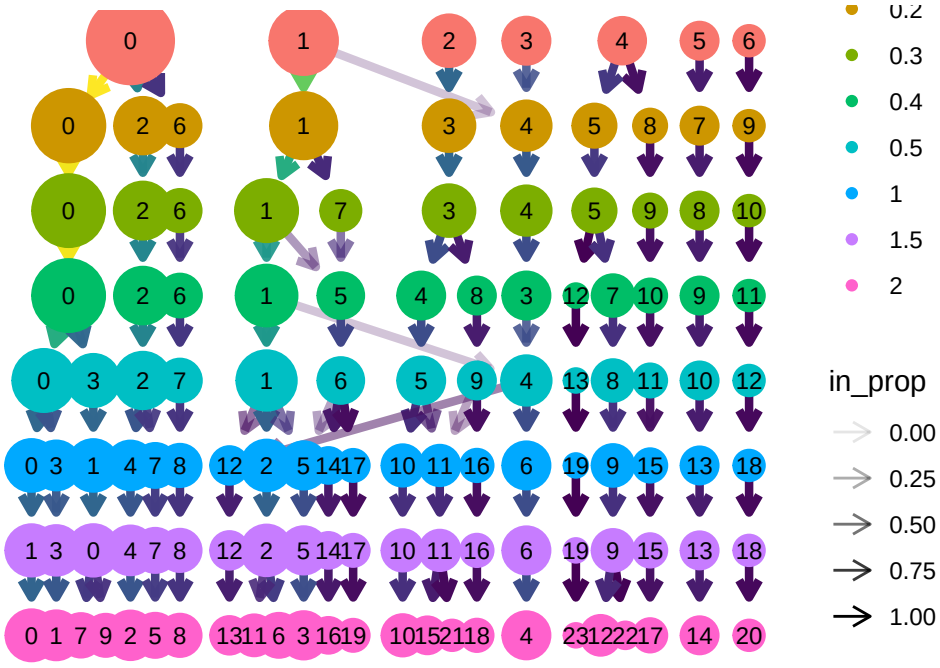


Figure 10.Figure 8. Clustree plot on clustering resolutions of 0.1, 0.2, 0.3, 0.4 and 0.5.

```
#choose your desired resolution:
p[[1]] + p[[2]] + p[[3]] + p[[4]]
```

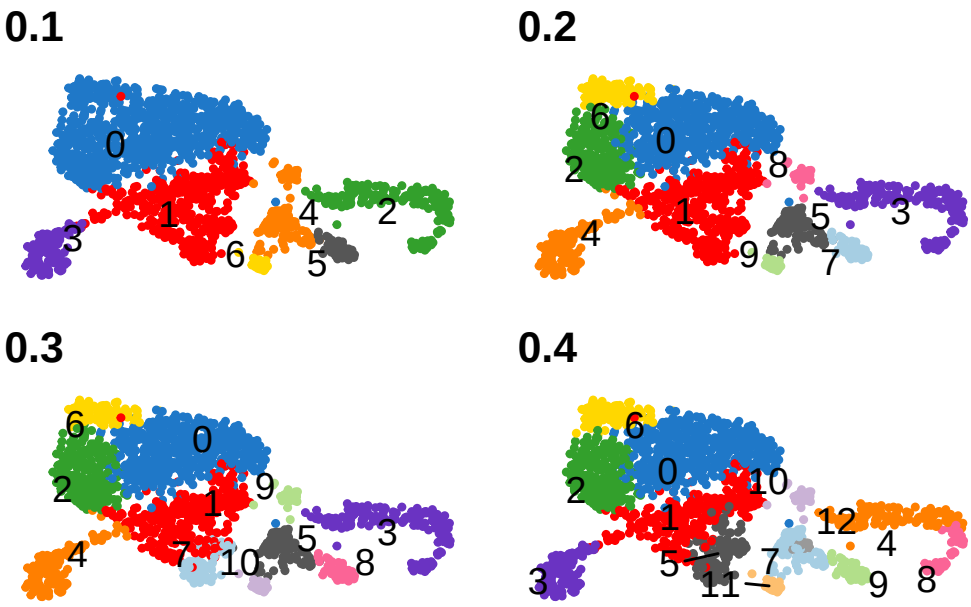


Figure 11. Cluster cell plots on resolutions 0.1, 0.2, 0.3 and 0.4. In the end, a resolution of 0.2 was considered optimal for clustering.

```
data3d <-FindClusters(object = data3d,
                      resolution = 0.2,
                      random.seed = 0)
```

```
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 1753
Number of edges: 34264

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9376
Number of communities: 10
Elapsed time: 0 seconds
```

```
data3d <- BuildClusterTree(
    object = data3d,
    reorder = TRUE,
    dims = d,
```

```
    reorder.numeric = T #Takes out 0 and starts at 1
  )
```

```
Reordering identity classes and rebuilding tree
```

```
ape::plot.phylo(data3d@tools$BuildClusterTree, direction = 'right')
```

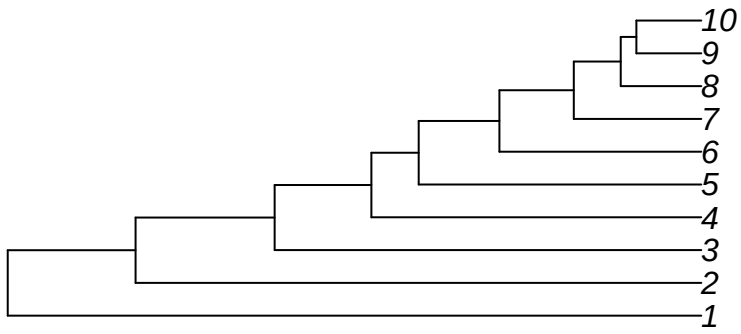


Figure 12. Phyloplot showing the relationships between clusters.

```
DimPlot(
    data3d,
    label = T,
    label.size = 5,
    repel = T,
    cols = clust.cp.separate
  ) &
    NoLegend() & NoAxes() &
    DimPlot(
      data3d,
      label = T,
```

```
    label.size = 5,
    repel = T,
    cols = clust.cp.separate,
    reduction = 'tsne'
  ) &
  NoLegend() & NoAxes()
```

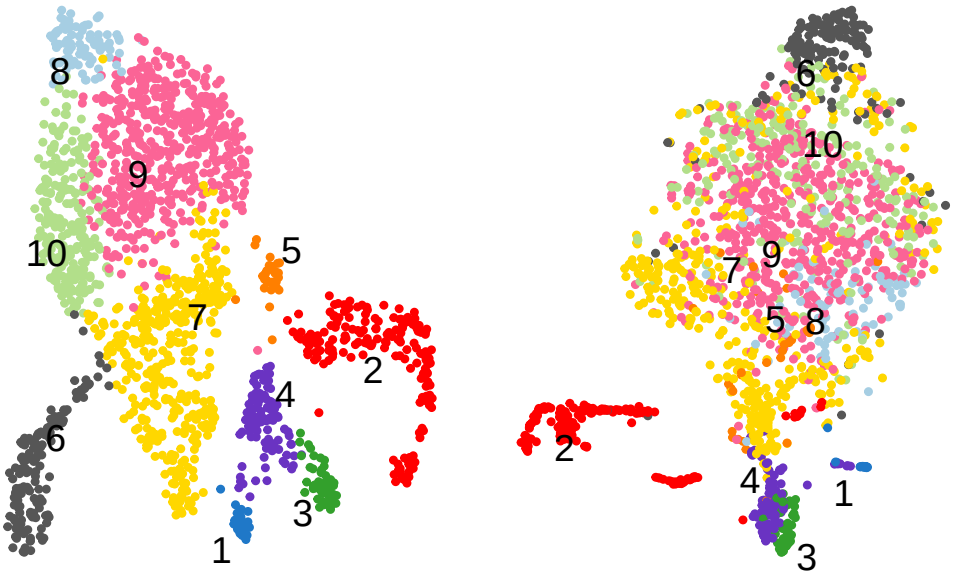


Figure 13. UMAP and tSNE cell plots with cells coloured according to cluster belonging. As cluster identification was performed on UMAP plots, it was expected clearer cluster boundaries when compared to tSNE plots.

Assign cluster names based on key expressed genes

```
##3.4 assign.cluster.names ----
  #* run a semi-automated script for updating cluster names based on marker gene expressio
  # you can generate a table to assign putative names to clusters:
clusterNames <- read_excel("/scratch/course/2023w300106/rochearcas/celltypemarkers.xlsx")
clusterNames$Marker[23]='Ncol3'
  #or you can build this in excel and import it:
  # clusterNames<- read_excel("Nv2.annotations.xlsx", sheet = 'cluster.annotations') #to u
goi = clusterNames$Marker
```

```
  #if cluster names are not integers: you tried this already and want to update / change i
data3d <- SetIdent(data3d, value = 'seurat_clusters')
data3d <- BuildClusterTree(data3d, reorder = T, reorder.numeric = T) #this re-sets everyth
```

```
Warning: The `slot` argument of `AverageExpression()` is deprecated as of Seurat 5.0.0.
i Please use the `layer` argument instead.
i The deprecated feature was likely used in the Seurat package.
  Please report the issue at <https://github.com/satijalab/seurat/issues>.
```

```
As of Seurat v5, we recommend using AggregateExpression to perform pseudo-bulk analysis.
First group.by variable `ident` starts with a number, appending `g` to ensure valid variable
Reordering identity classes and rebuilding tree
```

```
This message is displayed once per session.
```

```
DotPlot(data3d, features = goi) + RotatedAxis() + NoLegend() #checks that all your genes a
```

```
Warning: The following requested variables were not found: Tpm2, Ncol3
```

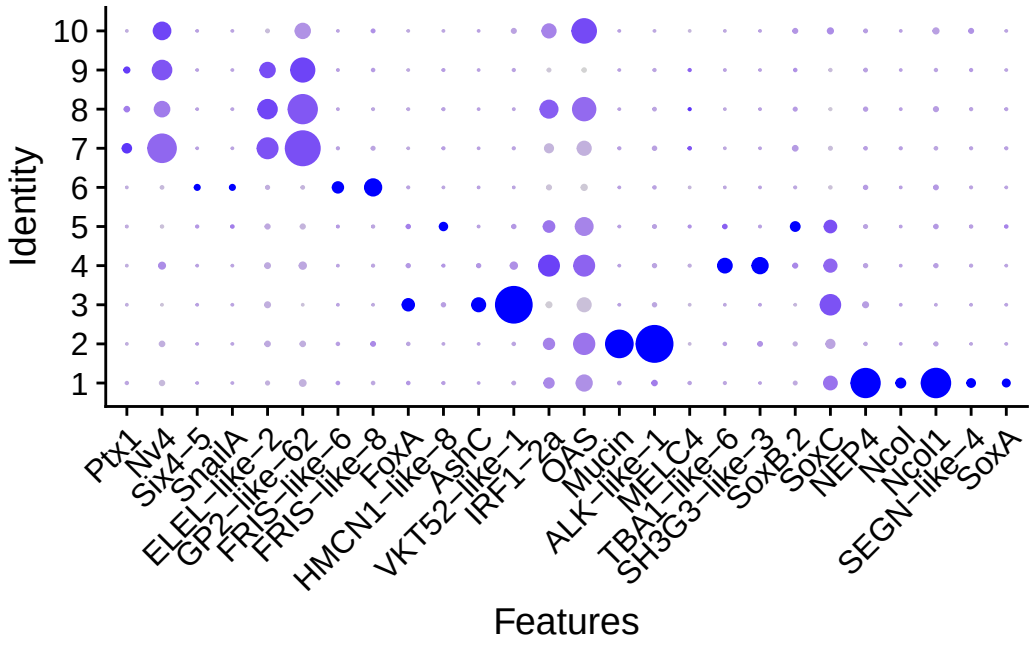
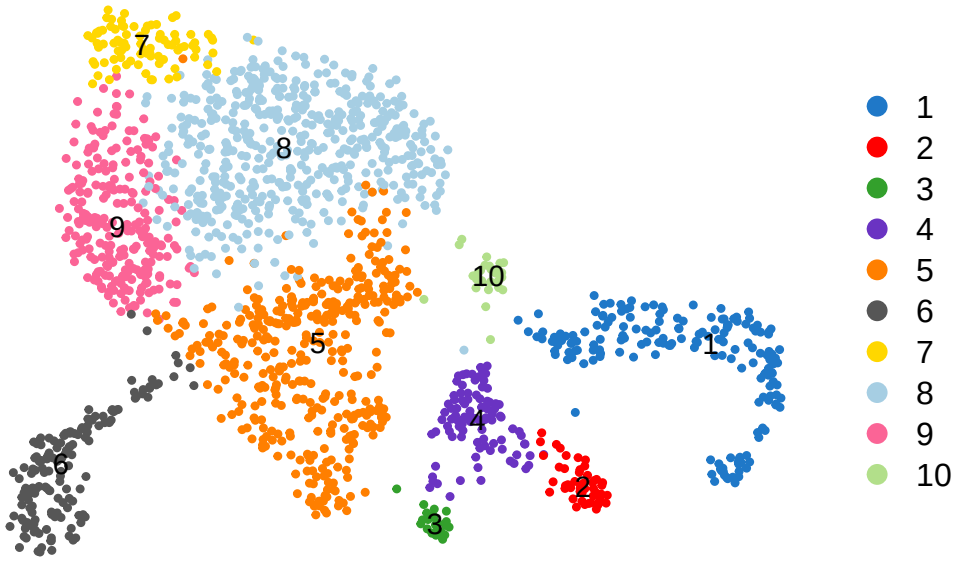


Figure 14. Cluster-specific expression of key genes that identify specific cell types. Each cluster was assigned a key expressed gene for identity tagging. An excel sheet correlates each gene with a particular cell type.

```
#plot your data so you can see which cluster was which:
DimPlot(data3d, label = T, cols = clust.cp.separate) + NoAxes()
```




```
  #assign cluster ID to the individual libraries:
```

Figure 15. UMAP cell plot with cells coloured according to cluster belonging. Each cluster receives an unique colour and number, which can be compared with the image above.

```
data3d <- ScaleData(data3d, features = goi, split.by = 'orig.ident') #works with scaled ge
```

```
Centering and scaling data matrix
```

```
Warning: Different features in new layer data than already exists for
scale.data
```

```
  #set empty name variable:
clName = vector()

  #generate a matrix of values of each cluster for each gene:
  #Seurat has a function for this
m = AverageExpression(data3d, features = goi, layer = 'scale.data')
```

```
Warning: The following 2 features were not found in the RNA assay: Tpm2, Ncol3
```

```
  # re-arranges the gene list in vs5: put it back in original order
gind=match(goi,rownames(m$RNA))

for (j in 1:length(levels(data3d@active.ident)))
    #for each cluster set
{
clName[j] = as.integer(which.max(m$RNA[, j])) #choose the highest value
}
sort(clName) #this prints which IDs were selected; sanity check that it did what you wante
```

```
 [1]  8  9 16 16 19 20 22 24 25 26
```

```
clusterNames$ID[clName] #you can also look at the names to see where the problem might hav
```

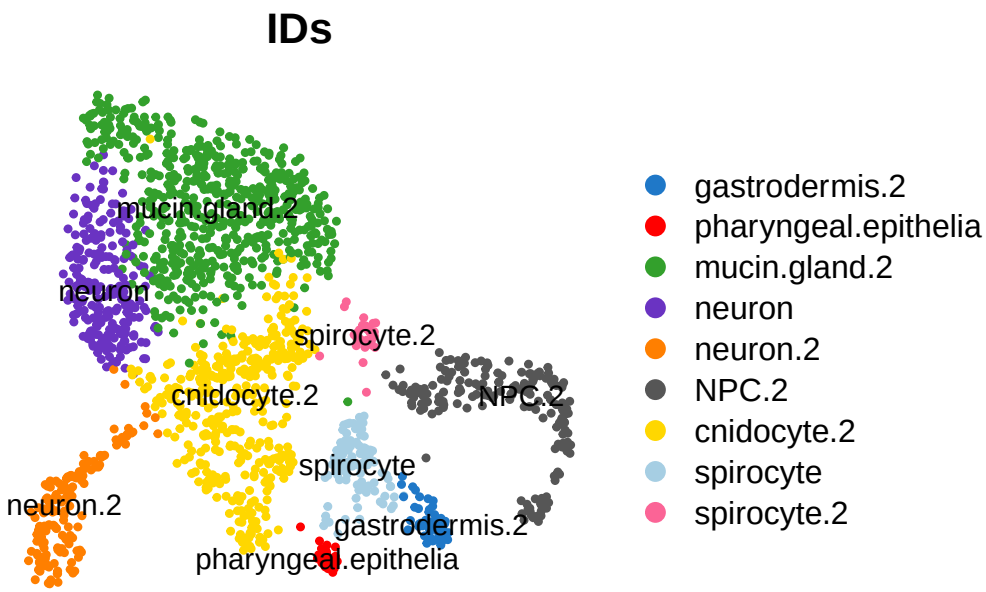
```
 [1] "NPC.2"               "gastrodermis.2"      "pharyngeal.epithelia"
 [4] "spirocyte"           "cnidocyte.2"         "neuron.2"
 [7] "mucin.gland.2"       "mucin.gland.2"       "neuron"
[10] "spirocyte.2"
```

```
  #first order the identities..
data3d@active.ident = factor(data3d@active.ident,
                             levels(data3d@active.ident)[order(clName)])
  #set the names to your IDs; once the gene list works, it is really easy to then update y
levels(data3d@active.ident) = clusterNames$ID[clName][order(clName)]

  #save the IDs in metadata:
data3d@meta.data$IDs = data3d@active.ident

  #plot to confirm all is as expected:
```

```
print(DimPlot(data3d,
          group.by = c('IDs'),
          label = T,
          cols = clust.cp.separate) + NoAxes())
```




```
data3d$orig.ident = droplevels(as.factor(data3d$orig.ident))
ids.cluster.sample = as.data.frame(table(Idents(data3d), data3d@meta.data$orig.ident))
colnames(ids.cluster.sample) = c('ID', 'sample', 'CellCount')

clust.cp = clust.cp.separate
  {
    #summarize the data of interest:
    data3d$orig.ident = droplevels(as.factor(data3d$orig.ident))
    ids.cluster.sample = as.data.frame(table(Idents(data3d), data3d@meta.data$orig.ident))
    colnames(ids.cluster.sample) = c('ID', 'sample', 'CellCount')

    #generate your sample and cluster UMAP plots:
    sample.plot =      DimPlot(
      data3d,
      group.by = 'orig.ident',
      order = rev(levels(data3d$orig.ident)),
```

```
  cols = rev(LibCP)
) + NoAxes() +
  labs(title = 'Time | sample origin')#+NoLegend()

cluster.plot.2 = DimPlot(
  data3d,
  label = T,
  label.size = 4,
  repel = T,
  reduction = 'umap',
  #order=rev(levels(data3d@active.ident)),
  #ordering is optional
  cols = clust.cp
) +
  # NoAxes() +
  labs(title = 'Clusters | ID') + NoLegend()
sample.plot+cluster.plot.2
#barplot of cluster identities in each sample:

dist.clust2 =
  ggplot(ids.cluster.sample, aes(fill = ID, y = CellCount,
                                 x = sample)) +
  geom_bar(
    mapping = aes(
      fill = ID,
      y = (CellCount),
      x = (sample)
    ),
    position = "fill",
    stat = "identity",
    width = 0.5
  ) +
  scale_fill_manual(values = clust.cp) +
  theme(axis.text.x = element_text(
    #face="bold", color="#993333",
    size = 8,
    angle = -45,
    hjust = 0,
    vjust = 0.5
  )) +
  geom_area(
```

```
      mapping = aes(
        fill = ID,
        y = (CellCount),
        x = as.integer(sample)
      ),
      position = "fill",
      stat = "identity",
      alpha = 0.2 ,
      linewidth = .5,
      colour = "white"
    ) +
    geom_bar(
      mapping = aes(
        fill = ID,
        y = (CellCount),
        #this re-plots the bars over the area
        x = (sample)
      ),
      position = "fill",
      stat = "identity",
      width = 0.5
    ) +
    ggtitle("Distribution of cell types in time and space")

}
#visualize your plots; can also do these individually as desired
print(dist.clust2 + cluster.plot.2)
```

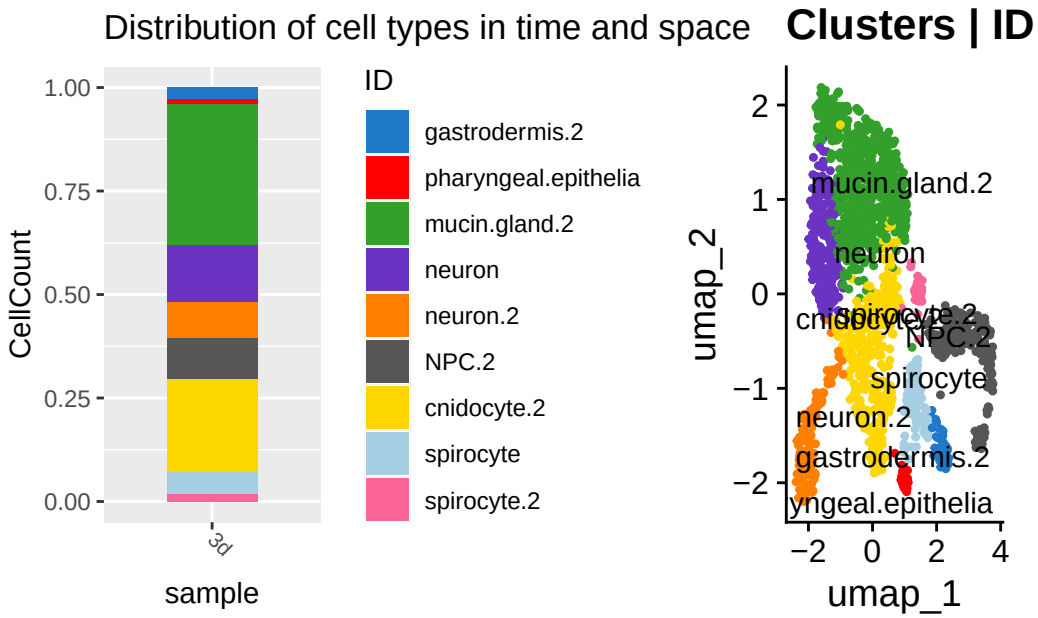


Figure 16. Left panel: Percentage of each cluster size, based on number of cells belonging to each cluster. Cellular identities come from key expressed genes. Right panel: UMAP cell plot, in which cell clusters are uniquely coloured and named according to identification to specific cell types of the 3-day-old Nematostella vectensis embryo.