celloracle 0.1.0 documentation

[celloracle](index.html#document-index)

* [Welcome to celloracle’s documentation!](index.html#document-index)
* [Installation](index.html#document-installation/index)
  + [Python Requirements](index.html#python-requirements)
    - [0. (Optional) Make a new environment](index.html#optional-make-a-new-environment)
    - [1. Add conda channels](index.html#add-conda-channels)
    - [2. Install velocyto](index.html#install-velocyto)
    - [3. Install scanpy](index.html#install-scanpy)
    - [4. Install gimmemotifs](index.html#install-gimmemotifs)
    - [5. Install other python libraries](index.html#install-other-python-libraries)
    - [6. install celloracle from github](index.html#install-celloracle-from-github)
  + [R requirements](index.html#r-requirements)
    - [Seurat](index.html#id3)
    - [Cicero](index.html#id5)
    - [igraph](index.html#id7)
    - [linkcomm](index.html#id9)
    - [rnetcarto](index.html#id11)
    - [Check installation](index.html#check-installation)
* [Tutorial](index.html#document-tutorials/index)
  + [1. ATAC-seq data preprocessing](index.html#document-tutorials/atac)
    - [A. Extract TF binding information from scATAC-seq data](index.html#a-extract-tf-binding-information-from-scatac-seq-data)
      * [0. Import library](index.html#document-notebooks/01_ATAC-seq_data_processing/option1_scATAC-seq_data_analysis_with_cicero/01_atacdata_to_cicero)
      * [1. Prepare data](index.html#1.-Prepare-data)
      * [2. Load data and make Cell Data Set (CDS) object](index.html#2.-Load-data-and-make-Cell-Data-Set-(CDS)-object)
        + [2.1. Process data to make CDS object](index.html#2.1.-Process-data-to-make-CDS-object)
      * [3. Qauality check and Filtering](index.html#3.-Qauality-check-and-Filtering)
      * [4. Process cicero-CDS object](index.html#4.-Process-cicero-CDS-object)
      * [5. Run cicero to get cis-regulatory connection scores](index.html#5.-Run-cicero-to-get-cis-regulatory-connection-scores)
      * [6. Save results for next step](index.html#6.-Save-results-for-next-step)
      * [0. Import libraries](index.html#document-notebooks/01_ATAC-seq_data_processing/option1_scATAC-seq_data_analysis_with_cicero/02_preprocess_peak_data)
      * [1. Load data made with cicero](index.html#1.-Load-data-made-with-cicero)
      * [2. Make TSS annotation](index.html#2.-Make-TSS-annotation)
      * [3. Integrate TSS info and cicero connections](index.html#3.-Integrate-TSS-info-and-cicero-connections)
      * [4. Filter peaks](index.html#4.-Filter-peaks)
      * [5. Save data](index.html#5.-Save-data)
    - [B. Extract TF binding information from bulk ATAC-seq data or Chip-seq data](index.html#b-extract-tf-binding-information-from-bulk-atac-seq-data-or-chip-seq-data)
      * [0. Import libraries](index.html#document-notebooks/01_ATAC-seq_data_processing/option2_Bulk_ATAC-seq_data/01_preprocess_Bulk_ATAC_seq_peak_data)
      * [1. Load bed file](index.html#1.-Load-bed-file)
      * [2. Make TSS annotation](index.html#2.-Make-TSS-annotation)
      * [3. Save data](index.html#3.-Save-data)
  + [2. Transcription factor binding motif scan](index.html#document-tutorials/motifscan)
    - [0. Import libraries](index.html#document-notebooks/02_motif_scan/02_atac_peaks_to_TFinfo_with_celloracle_190901)
    - [1. Load data](index.html#1.-Load-data)
    - [2. Check data](index.html#2.-Check-data)
      * [2.1. Remove short peaks](index.html#2.1.-Remove-short-peaks)
    - [3. Instantiate TFinfo object and search for TF binding motifs](index.html#3.-Instantiate-TFinfo-object-and-search-for-TF-binding-motifs)
    - [3.1 check reference genome installation](index.html#3.1-check-reference-genome-installation)
      * [3.2. Install reference genome (if refgenome is not installed)](index.html#3.2.-Install-reference-genome-(if-refgenome-is-not-installed))
    - [4. Scan motifs and save object](index.html#4.-Scan-motifs-and-save-object)
    - [5. Filtering motifs](index.html#5.-Filtering-motifs)
    - [6. Get Final results](index.html#6.-Get-Final-results)
      * [6.1. Get resutls as a dictionary](index.html#6.1.-Get-resutls-as-a-dictionary)
      * [6.2. Get results as a dataframe](index.html#6.2.-Get-results-as-a-dataframe)
    - [7. Save TFinfo as dictionary and dataframe](index.html#7.-Save-TFinfo-as-dictionary-and-dataframe)
  + [3. Single-cell RNA-seq data preprocessing](index.html#document-tutorials/scrnaprocess)
    - [A. scRNA-seq data preprocessing with scanpy](index.html#a-scrna-seq-data-preprocessing-with-scanpy)
      * [0. Import libraries](index.html#document-notebooks/03_scRNA-seq_data_preprocessing/scanpy_preprocessing_with_Paul_etal_2015_data)
      * [1. Load data](index.html#1.-Load-data)
      * [2. Filtering](index.html#2.-Filtering)
      * [3. Normalization](index.html#3.-Normalization)
      * [4. Variable gene detection](index.html#4.-Variable-gene-detection)
      * [5. Log transformation](index.html#5.-Log-transformation)
      * [6. Dimensional reduction](index.html#6.-Dimensional-reduction)
      * [7. Clustering](index.html#7.-Clustering)
      * [(Optional) Re-calculate Dimensional reduction graph](index.html#(Optional)-Re-calculate-Dimensional-reduction-graph)
      * [8. Check data](index.html#8.-Check-data)
        + [8.1. Visualize marker gene expression](index.html#8.1.-Visualize-marker-gene-expression)
      * [8. Make annotation for cluster](index.html#8.-Make-annotation-for-cluster)
        + [8.1. Make annotation (1)](index.html#8.1.-Make-annotation-(1))
        + [8.2. Make annotation (2)](index.html#8.2.-Make-annotation-(2))
      * [9. (Option) Select cell cluster](index.html#9.-(Option)-Select-cell-cluster)
      * [9. Save data](index.html#9.-Save-data)
    - [B. scRNA-seq data preprocessing with Seurat](index.html#b-scrna-seq-data-preprocessing-with-seurat)
  + [4. Network analysis](index.html#document-tutorials/networkanalysis)
    - [0. Import libraries](index.html#document-notebooks/04_Network_analysis/Network_analysis_with_with_Paul_etal_2015_data)
      * [0.1. Check installation](index.html#0.1.-Check-installation)
      * [0.2. Make a folder to save graph](index.html#0.2.-Make-a-folder-to-save-graph)
    - [1. Load data](index.html#1.-Load-data)
      * [1.1. Load processed gene expression data (anndata)](index.html#1.1.-Load-processed-gene-expression-data-(anndata))
      * [1.2. Load TF data.](index.html#1.2.-Load-TF-data.)
    - [2. Initiate Oracle object](index.html#2.-Initiate-Oracle-object)
      * [2.1. load gene expression data into oracle object.](index.html#2.1.-load-gene-expression-data-into-oracle-object.)
      * [2.2. load TFinfo into oracle object](index.html#2.2.-load-TFinfo-into-oracle-object)
      * [2.3. (Optional) Add TF info manually](index.html#2.3.-(Optional)-Add-TF-info-manually)
        + [2.3.1. Make TF info dictionary manually](index.html#2.3.1.-Make-TF-info-dictionary-manually)
        + [2.3.2. Add TF informatio dictionary into the oracle object](index.html#2.3.2.-Add-TF-informatio-dictionary-into-the-oracle-object)
    - [3. Knn imputation](index.html#3.-Knn-imputation)
      * [3.1. PCA](index.html#3.1.-PCA)
      * [3.2. Knn imputation](index.html#3.2.-Knn-imputation)
    - [4. Save and Load.](index.html#4.-Save-and-Load.)
    - [4. GRN calculation](index.html#4.-GRN-calculation)
    - [5. Network preprocessing](index.html#5.-Network-preprocessing)
      * [5.1. Filter network edges](index.html#5.1.-Filter-network-edges)
      * [5.2. Degree distribution](index.html#5.2.-Degree-distribution)
      * [5.3. Calculate netowrk score](index.html#5.3.-Calculate-netowrk-score)
      * [5.4. Save](index.html#5.4.-Save)
    - [6. Network analysis; Network score for each gene](index.html#6.-Network-analysis;-Network-score-for-each-gene)
      * [6.1. Network score in each cluster](index.html#6.1.-Network-score-in-each-cluster)
      * [6.2. Network score comparison between two clusters](index.html#6.2.-Network-score-comparison-between-two-clusters)
      * [6.3. Network score dynamics](index.html#6.3.-Network-score-dynamics)
      * [6.4. Gene cartography analysis](index.html#6.4.-Gene-cartography-analysis)
    - [7. Network analysis; Network score distribution](index.html#7.-Network-analysis;-Network-score-distribution)
      * [7.1. Distribution of network degree](index.html#7.1.-Distribution-of-network-degree)
      * [7.2. Distribution of netowrk entolopy](index.html#7.2.-Distribution-of-netowrk-entolopy)
  + [5. Simulation with GRNs](index.html#document-tutorials/simulation)
    - [0. Import libraries](index.html#document-notebooks/05_simulation/Gata1_KO_simulation_with_with_Paul_etal_2015_data)
      * [0.1. Import public libraries](index.html#0.1.-Import-public-libraries)
      * [0.1. Make a folder to save graph](index.html#0.1.-Make-a-folder-to-save-graph)
    - [1. Load data](index.html#1.-Load-data)
      * [1.1. Load processed oracle object](index.html#1.1.-Load-processed-oracle-object)
      * [1.2. Load inferred GRNs](index.html#1.2.-Load-inferred-GRNs)
    - [3. Make predictive models for simulation](index.html#3.-Make-predictive-models-for-simulation)
    - [4. in silico Perturbation-simulation](index.html#4.-in-silico-Perturbation-simulation)
      * [4.1. Check gene expression pattern.](index.html#4.1.-Check-gene-expression-pattern.)
      * [4.1. calculate future gene expression after perturbation.](index.html#4.1.-calculate-future-gene-expression-after-perturbation.)
      * [4.2. calculate transition probability between cells](index.html#4.2.-calculate-transition-probability-between-cells)
      * [4.3. Visualization](index.html#4.3.-Visualization)
        + [4.3.1. Detailed directed trajectory graph](index.html#4.3.1.-Detailed-directed-trajectory-graph)
        + [4.3.2. Grid graph](index.html#4.3.2.-Grid-graph)
      * [4.4. Markov simulation to analyze the effects of perturbation on cell fate transition](index.html#4.4.-Markov-simulation-to-analyze-the-effects-of-perturbation-on-cell-fate-transition)
        + [4.4.1. Do simulation](index.html#4.4.1.-Do-simulation)
        + [4.4.2. Check the results of simulation focusing on a specific cells](index.html#4.4.2.-Check-the-results-of-simulation-focusing-on-a-specific-cells)
        + [4.4.3. Summarize the results of simulation by plotting sankey diagram](index.html#4.4.3.-Summarize-the-results-of-simulation-by-plotting-sankey-diagram)
* [API](index.html#document-modules/index)
  + [Command Line API](index.html#command-line-api)
  + [Python API](index.html#python-api)
    - [Custom class in celloracle](index.html#document-modules/celloracle)
    - [Modules for ATAC-seq analysis](index.html#modules-for-atac-seq-analysis)
      * [celloracle.motif\_analysis module](index.html#document-modules/celloracle.motif_analysis)
    - [Modules for Network analysis](index.html#modules-for-network-analysis)
      * [celloracle.network\_analysis module](index.html#document-modules/celloracle.network_analysis)
    - [Other modules](index.html#other-modules)
      * [celloracle.go\_analysis module](index.html#document-modules/celloracle.go_analysis)
      * [celloracle.utility module](index.html#document-modules/celloracle.utility)
      * [celloracle.data module](index.html#document-modules/celloracle.data)
      * [celloracle.data\_conversion module](index.html#document-modules/celloracle.data_conversion)
* [Changelog](index.html#document-changelog)
* [License](index.html#document-license/index)
* [Authors and citations](index.html#document-citation/index)
  + [Cite celloracle](index.html#cite-celloracle)
  + [celloracle software development](index.html#celloracle-software-development)

[celloracle](index.html#document-index)

* [Docs](index.html#document-index) »
* celloracle 0.1.0 documentation

# Welcome to celloracle’s documentation![¶](#welcome-to-celloracle-s-documentation)

CellOracle is a python library for the analysis of Gene Regulatory Network with single cell data.

Source code and tutorials are available at [celloracle git hub repository](https://github.com/KenjiKamimoto-wustl122/CellOracle)

For colleagues. Thank you for your kind help!! Documentation is also available as word file and pdf file.

Could you please add comments on the files and send back it to Kenji? I appreciate your help!

[pdf version](_downloads/celloracle.pdf)

microsoft word version

Warning

CellOracle is still under development. It is alpha version and functions in this package may change in the future release.

# Contents[¶](#contents)

## Installation[¶](#installation)

celloracle use several python libraries and R library. Please follow this guide below to install dependent software of celloracle.

### Python Requirements[¶](#python-requirements)

* celloracle was developed with python 3.6. We do not support python 2.7x or python <=3.5.
* celloracle was depeloped in Linux and MacOS. We do not guarantee that celloracle works in in Windows OS.
* We highly recommend using [anaconda](https://www.continuum.io/downloads) to setup python environment.
* Please install all dependent libraries before installing celloracle according to the instructions below.
* celloracle itself is still beta version and it is not available through pypi or anaconda distribution yet. Please install celloracle from github repository according to the instruction below.

#### 0. (Optional) Make a new environment[¶](#optional-make-a-new-environment)

This step is optional. Please make a new virtual environment for celloracle and install dependent libraries in it if you get some software conflicts.

conda make -n celloracle\_env python=3.6  
conda activate celloracle\_env

#### 1. Add conda channels[¶](#add-conda-channels)

Some libraries below require non-default anaconda channels. Please add the channels below. Instead, you can explicitly enter the channel when you install a library.

conda config --add channels defaults  
conda config --add channels bioconda  
conda config --add channels conda-forge

#### 2. Install [velocyto](http://velocyto.org/velocyto.py/install/index.html)[¶](#install-velocyto)

Please install velocyto with the following commands or [the author’s instruction](http://velocyto.org/velocyto.py/install/index.html) . On Mac OS, you may have a compile error during velocyto installation. I recommend you installing [Xcode](https://developer.apple.com/xcode/) in that case.

conda install numpy scipy cython numba matplotlib scikit-learn h5py click

Then

pip install velocyto

#### 3. Install [scanpy](https://scanpy.readthedocs.io/en/stable/installation.html)[¶](#install-scanpy)

Please install scanpy with the following commands or [the author’s instruction](https://scanpy.readthedocs.io/en/stable/installation.html) .

conda install seaborn scikit-learn statsmodels numba pytables python-igraph louvain

Then

pip install scanpy

#### 4. Install [gimmemotifs](https://gimmemotifs.readthedocs.io/en/master/installation.html)[¶](#install-gimmemotifs)

Please install gimmemotifs with the following commands or [the author’s instruction](https://gimmemotifs.readthedocs.io/en/master/installation.html) .

conda install gimmemotifs

#### 5. Install other python libraries[¶](#install-other-python-libraries)

Please install other python libraries below with the following commands.

conda install goatools pyarrow tqdm joblib jupyter

#### 6. install celloracle from github[¶](#install-celloracle-from-github)

pip install git+https://github.com/KenjiKamimoto-wustl122/CellOracle

### R requirements[¶](#r-requirements)

celloracle use R library for the network analysis and scATAC-seq analysis. Please install [R](https://www.r-project.org) (>=3.5) and R libraries below according to the author’s instruction.

#### [Seurat](https://satijalab.org/seurat/install.html)[¶](#id3)

Please install seurat with the following r-script or [the author’s instruction](https://satijalab.org/seurat/install.html) . celloracle is compatible with both Seurat V2 and V3. If you use only scanpy for the scRNA-seq preprocessing and do not use Seurat, you can skip installation of Seurat.

in R console,

install.packages('Seurat')

#### [Cicero](https://cole-trapnell-lab.github.io/cicero-release/docs/#installing-cicero)[¶](#id5)

Please install Cicero with the following r-script or [the author’s instruction](https://cole-trapnell-lab.github.io/cicero-release/docs/#installing-cicero) . If you will not do scATAC-seq analysis and plan to run celloracle with default TF information which was builtin celloracle, you can skip installation of Cicero.

in R console,

if (!requireNamespace("BiocManager", quietly = TRUE))  
install.packages("BiocManager")  
BiocManager::install("cicero", version = "3.8")

#### [igraph](https://igraph.org/r/)[¶](#id7)

Please install igraph with the following r-script or [the author’s instruction](https://igraph.org/r/) .

in R console,

install.packages("igraph")

#### [linkcomm](https://cran.r-project.org/web/packages/linkcomm/index.html)[¶](#id9)

Please install linkcomm with the following r-script or [the author’s instruction](https://cran.r-project.org/web/packages/linkcomm/index.html) .

in R console,

install.packages("linkcomm")

#### [rnetcarto](https://github.com/cran/rnetcarto/blob/master/src/rgraph/README.md)[¶](#id11)

rnetcarto installation is little bit complicated. Please install rnetcarto with [the author’s instruction](https://github.com/cran/rnetcarto/blob/master/src/rgraph/README.md) . You also need to install [the GNU Scientific Libraries](https://www.gnu.org/software/gsl/) before installing rnetcarto. Detailed instruction can be found [here](https://github.com/cran/rnetcarto/blob/master/src/rgraph/README.md) .

#### Check installation[¶](#check-installation)

These R libraries above are necessary for the network analysis in celloracle. You can check installation using celloracle’s function.

in python console,

import celloracle as co  
co.network\_analysis.test\_R\_libraries\_installation()

Please make sure that all R libraries are installed. The following message will be shown when all R libraries are appropriately installed.

checking R library installation: gProfileR -> OK

checking R library installation: igraph -> OK

checking R library installation: linkcomm -> OK

checking R library installation: rnetcarto -> OK

## Tutorial[¶](#tutorial)

Analysis of celloracle proceeds through multiple steps below. Please run the notebooks following the order. If you do not have ATAC-seq data and want to use default genome and TF binding information, you can skip the first and second step and start from the third step. Please refer to the celloracle paper for detailed information.

You can download all notebooks from [here](https://github.com/KenjiKamimoto-wustl122/CellOracle/tree/master/docs/notebooks) .

### 1. ATAC-seq data preprocessing[¶](#atac-seq-data-preprocessing)

Process scATAC-seq data (or bulk ATAC-seq data) to get open accessible promoter/enhancer DNA sequence. If you have a scATAC-seq data, proximal and distal cis-regulatory elements from TSS site can be picked up using [Cicero](https://cole-trapnell-lab.github.io/cicero-release/docs/#installing-cicero) . If you have bulk ATAC-seq data, open accessible DNA elements around TSS site will be picked up.

#### A. Extract TF binding information from scATAC-seq data[¶](#a-extract-tf-binding-information-from-scatac-seq-data)

If you want to use scATAC-seq data, you can start from Cicero analysis to get information of distal cis-regulatory elements. This step use Cicero and do not use celloracle. Please refer to [the documentation of Cicero](https://cole-trapnell-lab.github.io/cicero-release/) for the detailed usage.

R notebook

##### 0. Import library[¶](#0.-Import-library)

[2]:

# install cicero before running this script  
rm(list=ls())  
library(cicero)

##### 1. Prepare data[¶](#1.-Prepare-data)

In this tutorial we use scAtac-seq data in 10x genomics database. You do not need to download these data when you analyze your scATAC data.

[4]:

# create folder to store data  
dir.create("data")  
  
# download demo dataset from 10x genomics  
system("wget -O data/matrix.tar.gz http://cf.10xgenomics.com/samples/cell-atac/1.1.0/atac\_v1\_E18\_brain\_fresh\_5k/atac\_v1\_E18\_brain\_fresh\_5k\_filtered\_peak\_bc\_matrix.tar.gz")  
  
# unzip data  
system("tar -xvf data/matrix.tar.gz -C data")

[6]:

# Just replace the data path below with the data path of your scATAC data if you have your scATAC data.  
  
data\_folder <- "data/filtered\_peak\_bc\_matrix"  
output\_folder <- "cicero\_output"  
dir.create(output\_folder)

##### 2. Load data and make Cell Data Set (CDS) object[¶](#2.-Load-data-and-make-Cell-Data-Set-(CDS)-object)

###### 2.1. Process data to make CDS object[¶](#2.1.-Process-data-to-make-CDS-object)

[7]:

# read in matrix data using the Matrix package  
indata <- Matrix::readMM(paste0(data\_folder, "/matrix.mtx"))  
# binarize the matrix  
indata@x[indata@x > 0] <- 1  
  
# format cell info  
cellinfo <- read.table(paste0(data\_folder, "/barcodes.tsv"))  
row.names(cellinfo) <- cellinfo$V1  
names(cellinfo) <- "cells"  
  
# format peak info  
peakinfo <- read.table(paste0(data\_folder, "/peaks.bed"))  
names(peakinfo) <- c("chr", "bp1", "bp2")  
peakinfo$site\_name <- paste(peakinfo$chr, peakinfo$bp1, peakinfo$bp2, sep="\_")  
row.names(peakinfo) <- peakinfo$site\_name  
  
row.names(indata) <- row.names(peakinfo)  
colnames(indata) <- row.names(cellinfo)  
  
# make CDS  
input\_cds <- suppressWarnings(newCellDataSet(indata,  
 phenoData = methods::new("AnnotatedDataFrame", data = cellinfo),  
 featureData = methods::new("AnnotatedDataFrame", data = peakinfo),  
 expressionFamily=VGAM::binomialff(),  
 lowerDetectionLimit=0))  
input\_cds@expressionFamily@vfamily <- "binomialff"  
input\_cds <- monocle::detectGenes(input\_cds)  
  
#Ensure there are no peaks included with zero reads  
input\_cds <- input\_cds[Matrix::rowSums(exprs(input\_cds)) >= 100,]

##### 3. Qauality check and Filtering[¶](#3.-Qauality-check-and-Filtering)

[8]:

# Visualize peak\_count\_per\_cell  
hist(Matrix::colSums(exprs(input\_cds)))

<_images/notebooks_01_ATAC-seq_data_processing_option1_scATAC-seq_data_analysis_with_cicero_01_atacdata_to_cicero_8_0.png>

[9]:

# filter cells by peak\_count  
max\_count <- 15000  
min\_count <- 2000  
input\_cds <- input\_cds[,Matrix::colSums(exprs(input\_cds)) >= min\_count]  
input\_cds <- input\_cds[,Matrix::colSums(exprs(input\_cds)) <= max\_count]

##### 4. Process cicero-CDS object[¶](#4.-Process-cicero-CDS-object)

[10]:

# Run cicero to get cis-regulatory networks  
set.seed(2017)  
input\_cds <- detectGenes(input\_cds)  
input\_cds <- estimateSizeFactors(input\_cds)  
  
input\_cds <- reduceDimension(input\_cds, max\_components = 2, verbose=T,scaling = FALSE,relative\_expr=FALSE,  
 reduction\_method = 'tSNE', norm\_method = "none")  
  
tsne\_coords <- t(reducedDimA(input\_cds))  
row.names(tsne\_coords) <- row.names(pData(input\_cds))  
cicero\_cds <- make\_cicero\_cds(input\_cds, reduced\_coordinates = tsne\_coords)  
  
saveRDS(cicero\_cds, paste0(output\_folder, "/cicero\_cds.Rds"))

Remove noise by PCA ...  
  
Reduce dimension by tSNE ...  
  
Overlap QC metrics:  
Cells per bin: 50  
Maximum shared cells bin-bin: 44  
Mean shared cells bin-bin: 0.76256263875674  
Median shared cells bin-bin: 0

##### 5. Run cicero to get cis-regulatory connection scores[¶](#5.-Run-cicero-to-get-cis-regulatory-connection-scores)

[11]:

# import genome length, which is needed for the function, run\_cicero  
mm10\_chromosome\_length <- read.table("./mm10\_chromosome\_length.txt")  
  
# runc the main function  
conns <- run\_cicero(cicero\_cds, mm10\_chromosome\_length) # Takes a few minutes to run  
  
# save results  
saveRDS(conns, paste0(output\_folder, "/cicero\_connections.Rds"))  
  
# check results  
head(conns)

[1] "Starting Cicero"  
[1] "Calculating distance\_parameter value"  
[1] "Running models"  
[1] "Assembling connections"  
[1] "Done"

A data.frame: 6 × 3

Peak1

Peak2

coaccess

<fct>

<fct>

<dbl>

2

chr1\_3094484\_3095479

chr1\_3113499\_3113979

-0.316289004

3

chr1\_3094484\_3095479

chr1\_3119478\_3121690

-0.419240532

4

chr1\_3094484\_3095479

chr1\_3399730\_3400368

-0.050867246

5

chr1\_3113499\_3113979

chr1\_3094484\_3095479

-0.316289004

7

chr1\_3113499\_3113979

chr1\_3119478\_3121690

0.370342744

8

chr1\_3113499\_3113979

chr1\_3399730\_3400368

-0.009276026

##### 6. Save results for next step[¶](#6.-Save-results-for-next-step)

[ ]:

all\_peaks <- row.names(exprs(input\_cds))  
write.csv(x = all\_peaks, file = paste0(output\_folder, "/all\_peaks.csv"))  
write.csv(x = conns, file = paste0(output\_folder, "/cicero\_connections.csv"))

Next, the results of Cicero analysis will be processed with celloracle to make TSS annotations.

Python notebook

In this notebook, we use the results of cicero analysis to get active promoter/enhancer DNA peaks. Cicero provides us information about the connection between peaks. First, we pick up peaks around Transcription starting site (TSS). Second, we merge cicero data with the peaks around TSS. Then we remove peaks that have a weak connection to TSS peak. The final product includes TSS peaks and peaks that have a strong connection with TSS peaks. We use this information as an active promoter/enhancer elements.

##### 0. Import libraries[¶](#0.-Import-libraries)

[1]:

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
import seaborn as sns  
#import time  
  
import os, sys, shutil, importlib, glob  
from tqdm import tqdm\_notebook as tqdm  
  
%config InlineBackend.figure\_format = 'retina'  
  
plt.rcParams['figure.figsize'] = (15,7)  
plt.rcParams["savefig.dpi"] = 200

[2]:

from celloracle import motif\_analysis as ma

##### 1. Load data made with cicero[¶](#1.-Load-data-made-with-cicero)

[3]:

# load all peaks  
peaks = pd.read\_csv("cicero\_output/all\_peaks.csv", index\_col=0)  
peaks = peaks.x.values  
peaks

[3]:

array(['chr1\_3094484\_3095479', 'chr1\_3113499\_3113979',  
 'chr1\_3119478\_3121690', ..., 'chrY\_90804622\_90805450',  
 'chrY\_90808626\_90809117', 'chrY\_90810560\_90811167'], dtype=object)

[4]:

# load cicero results  
cicero\_connections = pd.read\_csv("cicero\_output/cicero\_connections.csv", index\_col=0)  
cicero\_connections.head()

/home/k/anaconda3/envs/test/lib/python3.6/site-packages/numpy/lib/arraysetops.py:568: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison  
 mask |= (ar1 == a)

[4]:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Peak1 | Peak2 | coaccess |
| 2 | chr1\_3094484\_3095479 | chr1\_3113499\_3113979 | -0.316289 |
| 3 | chr1\_3094484\_3095479 | chr1\_3119478\_3121690 | -0.419241 |
| 4 | chr1\_3094484\_3095479 | chr1\_3399730\_3400368 | -0.050867 |
| 5 | chr1\_3113499\_3113979 | chr1\_3094484\_3095479 | -0.316289 |
| 7 | chr1\_3113499\_3113979 | chr1\_3119478\_3121690 | 0.370343 |

##### 2. Make TSS annotation[¶](#2.-Make-TSS-annotation)

!! Please make sure that you are setting correct ref genoms.

[5]:

tss\_annotated = ma.get\_tss\_info(peak\_str\_list=peaks, ref\_genome="mm10")  
  
# check results  
tss\_annotated.tail()

que bed peaks: 72402  
tss peaks in que: 16987

[5]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | chr | start | end | gene\_short\_name | strand |
| 16982 | chr1 | 55130650 | 55132118 | Mob4 | + |
| 16983 | chr6 | 94499875 | 94500767 | Slc25a26 | + |
| 16984 | chr19 | 45659222 | 45660823 | Fbxw4 | - |
| 16985 | chr12 | 100898848 | 100899597 | Gpr68 | - |
| 16986 | chr4 | 129491262 | 129492047 | Fam229a | - |

##### 3. Integrate TSS info and cicero connections[¶](#3.-Integrate-TSS-info-and-cicero-connections)

The output file after integrartion process have three columns; peak\_id, gene\_short\_name, coaccess. “peak\_id” is either of TSS peak or peaks that have cicero connection between TSS peak. “gene\_short\_name” is gene name of TSS site. “coaccess” is a coacces score between TSS peak. If the score is 1, it means that the peak\_id is TSS itself.

[8]:

integrated = ma.integrate\_tss\_peak\_with\_cicero(tss\_peak=tss\_annotated,  
 cicero\_connections=cicero\_connections)  
print(integrated.shape)  
integrated.head()

(263279, 3)

[8]:

|  |  |  |  |
| --- | --- | --- | --- |
|  | peak\_id | gene\_short\_name | coaccess |
| 0 | chr10\_100015291\_100017830 | Kitl | 1.000000 |
| 1 | chr10\_100018677\_100020384 | Kitl | 0.086299 |
| 2 | chr10\_100050858\_100051762 | Kitl | 0.034558 |
| 3 | chr10\_100052829\_100053395 | Kitl | 0.167188 |
| 4 | chr10\_100128086\_100128882 | Tmtc3 | 0.022341 |

##### 4. Filter peaks[¶](#4.-Filter-peaks)

Remove peaks that have weak coaccess score.

[9]:

peak = integrated[integrated.coaccess >= 0.8]  
peak = peak[["peak\_id", "gene\_short\_name"]].reset\_index(drop=True)

[10]:

print(peak.shape)  
peak.head()

(15680, 2)

[10]:

|  |  |  |
| --- | --- | --- |
|  | peak\_id | gene\_short\_name |
| 0 | chr10\_100015291\_100017830 | Kitl |
| 1 | chr10\_100486534\_100488209 | Tmtc3 |
| 2 | chr10\_100588641\_100589556 | 4930430F08Rik |
| 3 | chr10\_100741247\_100742505 | Gm35722 |
| 4 | chr10\_101681379\_101682124 | Mgat4c |

##### 5. Save data[¶](#5.-Save-data)

Save the promoter/enhancer peak.

[11]:

peak.to\_parquet("peak\_file.parquet")

-> go to next notebook

#### B. Extract TF binding information from bulk ATAC-seq data or Chip-seq data[¶](#b-extract-tf-binding-information-from-bulk-atac-seq-data-or-chip-seq-data)

Instead of scATAC-seq data bulk DNA-seq data can be used.

Python notebook

##### 0. Import libraries[¶](#0.-Import-libraries)

[1]:

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
import seaborn as sns  
#import time  
  
import os, sys, shutil, importlib, glob  
from tqdm import tqdm\_notebook as tqdm  
  
%config InlineBackend.figure\_format = 'retina'  
  
plt.rcParams['figure.figsize'] = (15,7)  
plt.rcParams["savefig.dpi"] = 200

[2]:

# import celloracle function  
from celloracle import motif\_analysis as ma

##### 1. Load bed file[¶](#1.-Load-bed-file)

Import bed file of ATAC-seq data. This script can also be used DNase-seq or Chip-seq data.

[3]:

file\_path\_of\_bed\_file = "data/all\_peaks.bed"

[4]:

# load bed\_file  
bed = ma.read\_bed(file\_path\_of\_bed\_file)  
print(bed.shape)  
bed.head()

(436206, 4)

[4]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | chrom | start | end | seqname |
| 0 | chr1 | 3002478 | 3002968 | chr1\_3002478\_3002968 |
| 1 | chr1 | 3084739 | 3085712 | chr1\_3084739\_3085712 |
| 2 | chr1 | 3103576 | 3104022 | chr1\_3103576\_3104022 |
| 3 | chr1 | 3106871 | 3107210 | chr1\_3106871\_3107210 |
| 4 | chr1 | 3108932 | 3109158 | chr1\_3108932\_3109158 |

[6]:

# convert bed file into peak name list  
peaks = ma.process\_bed\_file.df\_to\_list\_peakstr(bed)  
peaks

[6]:

array(['chr1\_3002478\_3002968', 'chr1\_3084739\_3085712',  
 'chr1\_3103576\_3104022', ..., 'chrY\_631222\_631480',  
 'chrY\_795887\_796426', 'chrY\_2397419\_2397628'], dtype=object)

##### 2. Make TSS annotation[¶](#2.-Make-TSS-annotation)

!! Please make sure that you are setting correct ref genoms.

[7]:

tss\_annotated = ma.get\_tss\_info(peak\_str\_list=peaks, ref\_genome="mm9")  
  
# check results  
tss\_annotated.tail()

que bed peaks: 436206  
tss peaks in que: 24822

[7]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | chr | start | end | gene\_short\_name | strand |
| 24817 | chr2 | 60560211 | 60561602 | Itgb6 | - |
| 24818 | chr15 | 3975177 | 3978654 | BC037032 | - |
| 24819 | chr14 | 67690701 | 67692101 | Ppp2r2a | - |
| 24820 | chr17 | 48455247 | 48455773 | B430306N03Rik | + |
| 24821 | chr10 | 59861192 | 59861608 | Gm17455 | + |

[9]:

# change format  
peak\_id\_tss = ma.process\_bed\_file.df\_to\_list\_peakstr(tss\_annotated)  
tss\_annotated = pd.DataFrame({"peak\_id": peak\_id\_tss,  
 "gene\_short\_name": tss\_annotated.gene\_short\_name.values})  
tss\_annotated = tss\_annotated.reset\_index(drop=True)  
print(tss\_annotated.shape)  
tss\_annotated.head()

(24822, 2)

[9]:

|  |  |  |
| --- | --- | --- |
|  | peak\_id | gene\_short\_name |
| 0 | chr7\_50691730\_50692032 | Nkg7 |
| 1 | chr7\_50692077\_50692785 | Nkg7 |
| 2 | chr13\_93564413\_93564836 | Thbs4 |
| 3 | chr13\_14613429\_14615645 | Hecw1 |
| 4 | chr3\_99688753\_99689665 | Spag17 |

##### 3. Save data[¶](#3.-Save-data)

[10]:

tss\_annotated.to\_parquet("peak\_file.parquet")

-> go to next notebook

### 2. Transcription factor binding motif scan[¶](#transcription-factor-binding-motif-scan)

Promoter/enhancer DNA sequence will be subjected to TF-binding motif scan to get the list of potential regulatory genes for each target gene.

Python notebook

#### 0. Import libraries[¶](#0.-Import-libraries)

[1]:

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
import seaborn as sns  
#import time  
  
import os, sys, shutil, importlib, glob  
from tqdm import tqdm\_notebook as tqdm  
  
%config InlineBackend.figure\_format = 'retina'  
  
plt.rcParams['figure.figsize'] = (15,7)  
plt.rcParams["savefig.dpi"] = 600

[3]:

from celloracle import motif\_analysis as ma  
from celloracle.utility import save\_as\_pickled\_object

#### 1. Load data[¶](#1.-Load-data)

load annotated peak data.

[4]:

peaks = pd.read\_parquet("../01\_ATAC-seq\_data\_processing/option1\_scATAC-seq\_data\_analysis\_with\_cicero/peak\_file.parquet")  
peaks.head()

[4]:

|  |  |  |
| --- | --- | --- |
|  | peak\_id | gene\_short\_name |
| 0 | chr10\_100015291\_100017830 | Kitl |
| 1 | chr10\_100486534\_100488209 | Tmtc3 |
| 2 | chr10\_100588641\_100589556 | 4930430F08Rik |
| 3 | chr10\_100741247\_100742505 | Gm35722 |
| 4 | chr10\_101681379\_101682124 | Mgat4c |

#### 2. Check data[¶](#2.-Check-data)

[5]:

# check data  
print(f"number of peak: {len(peaks.peak\_id.unique())}")  
mean\_ = len(peaks.groupby(["gene\_short\_name", "peak\_id"]).count())/len(peaks.gene\_short\_name.unique())  
print(f"mean peaks per gene: {mean\_}")  
  
def getLength(x):  
 a, b, c = x["peak\_id"].split("\_")  
 return int(c) - int(b)  
  
#  
df = peaks.apply(lambda x: getLength(x), axis=1)  
print(f"mean peak length: {df.values.mean()}")

number of peak: 13919  
mean peaks per gene: 1.0591731964333964  
mean peak length: 1756.1744260204082

##### 2.1. Remove short peaks[¶](#2.1.-Remove-short-peaks)

Short peak cannot be used for motif scan. Remove short DNA fragment.

[6]:

peaks = peaks[df>=5]

#### 3. Instantiate TFinfo object and search for TF binding motifs[¶](#3.-Instantiate-TFinfo-object-and-search-for-TF-binding-motifs)

In celloracle motif analysis, we use custom class; TFinfo. This class convert peak data into get DNA sequence and scan the DNA sequence searching for TF binding motifs. Then the results of motif scan will be filtered and converted into several files, such as python dictionary or dataframe. These TF information is necessary for celloracle’s GRN inference.

#### 3.1 check reference genome installation[¶](#3.1-check-reference-genome-installation)

[7]:

# PLEASE make sure that you are setting correct ref genome.  
ref\_genome = "mm10"  
  
ma.is\_genome\_installed(ref\_genome=ref\_genome)

genome mm10 is not installed in this environment.  
Please install genome using genomepy.  
e.g.  
 >>> import genomepy  
 >>> genomepy.install\_genome("mm9", "UCSC")

[7]:

False

##### 3.2. Install reference genome (if refgenome is not installed)[¶](#3.2.-Install-reference-genome-(if-refgenome-is-not-installed))

[9]:

import genomepy  
genomepy.install\_genome(ref\_genome, "UCSC")

downloading from http://hgdownload.soe.ucsc.edu/goldenPath/mm10/bigZips/chromFa.tar.gz...  
done...  
name: mm10  
local name: mm10  
fasta: /home/k/.local/share/genomes/mm10/mm10.fa

[9]:

# check again  
ma.is\_genome\_installed(ref\_genome=ref\_genome)

[9]:

True

[14]:

# Instantiate TFinfo object  
tfi = ma.TFinfo(peak\_data\_frame=peaks, # peak info calculated from ATAC-seq data  
 ref\_genome=ref\_genome)

#### 4. Scan motifs and save object[¶](#4.-Scan-motifs-and-save-object)

This step requires computational resource and may take long time

[15]:

%%time  
# Scan motifs  
tfi.scan(fpr=0.02, verbose=True)  
  
# sace tfinfo object  
tfi.to\_hdf5(file\_path="test.celloracle.tfinfo")

initiating scanner ...

2019-09-22 23:00:18,604 - INFO - Using background: genome mm10 with length 200  
2019-09-22 23:00:18,986 - INFO - Determining FPR-based threshold

getting DNA sequences ...  
scanning motifs ...

CPU times: user 52min 23s, sys: 36.8 s, total: 53min  
Wall time: 52min 58s

[16]:

# check motif scan results  
tfi.scanned\_df.head()

[16]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | seqname | motif\_id | factors\_direct | factors\_indirect | score | pos | strand |
| 0 | chr10\_100015291\_100017830 | GM.5.0.Homeodomain.0001 | TGIF1 | ENSG00000234254, TGIF1 | 10.311002 | 1003 | 1 |
| 1 | chr10\_100015291\_100017830 | GM.5.0.Mixed.0001 |  | SRF, EGR1 | 7.925873 | 481 | 1 |
| 2 | chr10\_100015291\_100017830 | GM.5.0.Mixed.0001 |  | SRF, EGR1 | 7.321375 | 911 | -1 |
| 3 | chr10\_100015291\_100017830 | GM.5.0.Mixed.0001 |  | SRF, EGR1 | 7.276585 | 811 | -1 |
| 4 | chr10\_100015291\_100017830 | GM.5.0.Nuclear\_receptor.0002 | NR2C2 | NR2C2, Nr2c2 | 9.067331 | 449 | -1 |

We have the score for each sequence and motif\_id pair. In the next step we will filter the motifs with low score.

#### 5. Filtering motifs[¶](#5.-Filtering-motifs)

[17]:

# reset filtering  
tfi.reset\_filtering()  
  
# do filtering  
tfi.filter\_motifs\_by\_score(threshold=10.5)  
  
# do post filtering process. Convert results into several file format.  
tfi.make\_TFinfo\_dataframe\_and\_dictionary(verbose=True)

peaks were filtered: 12934005 -> 2285279  
1. converting scanned results into one-hot encoded dataframe.

2. converting results into dictionaries.  
converting scan results into dictionaries...

#### 6. Get Final results[¶](#6.-Get-Final-results)

##### 6.1. Get resutls as a dictionary[¶](#6.1.-Get-resutls-as-a-dictionary)

[18]:

td = tfi.to\_dictionary(dictionary\_type="targetgene2TFs")

##### 6.2. Get results as a dataframe[¶](#6.2.-Get-results-as-a-dataframe)

[20]:

df = tfi.to\_dataframe()  
df.head()

[20]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | peak\_id | gene\_short\_name | 9430076c15rik | Ac002126.6 | Ac012531.1 | Ac226150.2 | Afp | Ahr | Ahrr | Aire | ... | Znf784 | Znf8 | Znf816 | Znf85 | Zscan10 | Zscan16 | Zscan22 | Zscan26 | Zscan31 | Zscan4 |
| 0 | chr10\_100015291\_100017830 | Kitl | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | chr10\_100486534\_100488209 | Tmtc3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | chr10\_100588641\_100589556 | 4930430F08Rik | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | chr10\_100741247\_100742505 | Gm35722 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | chr10\_101681379\_101682124 | Mgat4c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

5 rows × 1092 columns

#### 7. Save TFinfo as dictionary and dataframe[¶](#7.-Save-TFinfo-as-dictionary-and-dataframe)

we’ll use this information to make GRN. Save this results.

[21]:

folder = "TFinfo\_outputs"  
os.makedirs(folder, exist\_ok=True)  
  
# save TFinfo as a dictionary  
td = tfi.to\_dictionary(dictionary\_type="targetgene2TFs")  
save\_as\_pickled\_object(td, os.path.join(folder, "TFinfo\_targetgene2TFs.pickled"))  
  
# save TFinfo as a dataframe  
df = tfi.to\_dataframe()  
df.to\_parquet(os.path.join(folder, "TFinfo\_dataframe.parquet"))

[ ]:

### 3. Single-cell RNA-seq data preprocessing[¶](#single-cell-rna-seq-data-preprocessing)

Network analysis and simulation in celloracle require processed scRNA-seq data. We recommend processing scRNA-seq data using either scanpy or Seurat. If you want to know general idea of scRNA-seq data processing, please go to [the documentation of scanpy](https://scanpy.readthedocs.io/en/stable/) and [the documentation of Seurat](https://satijalab.org/seurat/vignettes.html) . This step needs to include data quality check, filtering, normalization, dimensional reduction, clustering and cluster annotation. If you have preprocessed scRNA-seq data, such as anndata or seurat object, you can skip this part.

Note

several tips

#### A. scRNA-seq data preprocessing with scanpy[¶](#a-scrna-seq-data-preprocessing-with-scanpy)

scanpy is a python library for the analysis of scRNA-seq data.

In this tutorial, we use the almost same code as [one of scanpy tutorials](https://scanpy-tutorials.readthedocs.io/en/latest/paga-paul15.html) with some modification.

Python notebook

##### 0. Import libraries[¶](#0.-Import-libraries)

[1]:

import os  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import scanpy as sc

[2]:

%matplotlib inline  
%config InlineBackend.figure\_format = 'retina'  
plt.rcParams["savefig.dpi"] = 300  
plt.rcParams["figure.figsize"] = [12,6]

##### 1. Load data[¶](#1.-Load-data)

In this notebook, we show an example of scRNA-seq preprocessing using scRNA-seq data of hematopoiesis (Paul, F., Arkin, Y., Giladi, A., Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., et al. (2015). Transcriptional Heterogeneity and Lineage Commitment in Myeloid Progenitors. Cell, 163(7), 1663–1677. <http://doi.org/10.1016/j.cell.2015.11.013>). This data can be download using a scanpy’s function.

Please change the code below if you want to use your data.

[3]:

# download dataset. You can change the code blow if you use another data.  
adata = sc.datasets.paul15()

WARNING: In Scanpy 0.\*, this returned logarithmized data. Now it returns non-logarithmized data.

... storing 'paul15\_clusters' as categorical  
Trying to set attribute `.uns` of view, making a copy.

##### 2. Filtering[¶](#2.-Filtering)

[4]:

# only consider genes with more than 1 count  
sc.pp.filter\_genes(adata, min\_counts=1)

##### 3. Normalization[¶](#3.-Normalization)

[5]:

# normalize gene expression matrix with total UMI count per cell  
sc.pp.normalize\_per\_cell(adata, key\_n\_counts='n\_counts\_all')

##### 4. Variable gene detection[¶](#4.-Variable-gene-detection)

In celloracle we will make GRN and do simulation using many genes. Removing weakly-variable gene genes will reduce calculation time. We recomend using top 2000~3000 variable genes.

[6]:

# select top2000 highly-variable genes  
filter\_result = sc.pp.filter\_genes\_dispersion(adata.X,  
 flavor='cell\_ranger',  
 n\_top\_genes=2000,  
 log=False)  
  
# subset the genes  
adata = adata[:, filter\_result.gene\_subset]  
  
# renormalize after filtering  
sc.pp.normalize\_per\_cell(adata)

Trying to set attribute `.obs` of view, making a copy.

##### 5. Log transformation[¶](#5.-Log-transformation)

Log transformation and scaling are necessary for many single cell analysis. But we still use non-transformed gene expression data in the celloracle analysis. We keep raw count in anndata with following command.

[7]:

# keep raw cont data before log transformation  
adata.raw = adata  
  
# Log transformation and scaling  
sc.pp.log1p(adata)  
sc.pp.scale(adata)

[8]:

# (optional) Regressing out some values help reduce batch effects.  
  
#sc.pp.regress\_out(adata, ['n\_counts', 'percent\_mito'])  
#sc.pp.regress\_out(adata, ['n\_counts'])

##### 6. Dimensional reduction[¶](#6.-Dimensional-reduction)

Dimensional reduction is one of the most important parts in the scRNA-seq analysis. In celloracle, we use dimensional reduction embeddings for the simulation of cell transition. We need to select an algorithm for dimensional reduction so that cell-cell relationship is properly represented. We recommend using one of the dimensional reduction algorithms (or trajectory inference algorithms); UMAP, tSNE, diffusion map, Force-directed graph drawing, PAGA.

In this example, we use a combination of diffusion, force-directed graph drawing, and PAGA.

[9]:

# PCA  
sc.tl.pca(adata, svd\_solver='arpack')

[10]:

# diffusion map  
sc.pp.neighbors(adata, n\_neighbors=4, n\_pcs=20)  
  
sc.tl.diffmap(adata)  
# calculate neihbors again based on diffusionmap  
sc.pp.neighbors(adata, n\_neighbors=10, use\_rep='X\_diffmap')

##### 7. Clustering[¶](#7.-Clustering)

[11]:

sc.tl.louvain(adata, resolution=0.8)

##### (Optional) Re-calculate Dimensional reduction graph[¶](#(Optional)-Re-calculate-Dimensional-reduction-graph)

[12]:

# PAGA graph construction  
sc.tl.paga(adata, groups='louvain')

[13]:

# check current cluster name  
cluster\_list = adata.obs.louvain.unique()  
cluster\_list

[13]:

[5, 2, 12, 13, 0, ..., 6, 20, 14, 15, 21]  
Length: 23  
Categories (23, object): [5, 2, 12, 13, ..., 20, 14, 15, 21]

[14]:

sc.pl.paga(adata)

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_22_0.png>

[15]:

sc.tl.draw\_graph(adata, init\_pos='paga', random\_state=123)

[16]:

sc.pl.draw\_graph(adata, color=['louvain'],  
 legend\_loc='on data')

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_24_0.png>

##### 8. Check data[¶](#8.-Check-data)

###### 8.1. Visualize marker gene expression[¶](#8.1.-Visualize-marker-gene-expression)

[17]:

markers = {"Erythroids":["Gata1", "Klf1", "Gypa", "Hba-a2"],  
 "Megakaryocytes":["Itga2b", "Pbx1", "Sdpr", "Vwf"],  
 "Granulocytes":["Elane", "Cebpe", "Ctsg", "Mpo", "Gfi1"],  
 "Monocytes":["Irf8", "Csf1r", "Ctsg", "Mpo"],  
 "Mast\_cells":["Cma1", "Gzmb", "Kit"],  
 "Basophils":["Mcpt8", "Prss34"]  
 }  
  
for cell\_type, genes in markers.items():  
 print(f"marker gene of {cell\_type}")  
 sc.pl.draw\_graph(adata, color=genes, use\_raw=False)  
 plt.show()

marker gene of Erythroids

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_26_1.png>

marker gene of Megakaryocytes

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_26_3.png>

marker gene of Granulocytes

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_26_5.png>

marker gene of Monocytes

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_26_7.png>

marker gene of Mast\_cells

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_26_9.png>

marker gene of Basophils

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_26_11.png>

##### 8. Make annotation for cluster[¶](#8.-Make-annotation-for-cluster)

Based on the marker gene expression and previous report, we will manually make annotation for each cluster.

###### 8.1. Make annotation (1)[¶](#8.1.-Make-annotation-(1))

[20]:

sc.pl.draw\_graph(adata, color=['louvain', 'paul15\_clusters'],  
 legend\_loc='on data')

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_30_0.png>

[21]:

# check current cluster name  
cluster\_list = adata.obs.louvain.unique()  
cluster\_list

[21]:

[5, 2, 12, 13, 0, ..., 6, 20, 14, 15, 21]  
Length: 23  
Categories (23, object): [5, 2, 12, 13, ..., 20, 14, 15, 21]

[22]:

# make anottation dictionary  
annotation = {"MEP":[5],  
 "Erythroids": [15, 10, 16, 9, 8, 14, 19, 3, 12, 18],  
 "Megakaryocytes":[17, 22],  
 "GMP":[11, 1],  
 "late\_GMP" :[0],  
 "Granulocytes":[7, 13, 4],  
 "Monocytes":[6, 2],  
 "DC":[21],  
 "Lymphoid":[20]}  
  
# change dictionary format  
annotation\_rev = {}  
for i in cluster\_list:  
 for k in annotation:  
 if int(i) in annotation[k]:  
 annotation\_rev[i] = k  
  
# check dictionary  
annotation\_rev

[22]:

{'5': 'MEP',  
 '2': 'Monocytes',  
 '12': 'Erythroids',  
 '13': 'Granulocytes',  
 '0': 'late\_GMP',  
 '10': 'Erythroids',  
 '3': 'Erythroids',  
 '18': 'Erythroids',  
 '11': 'GMP',  
 '7': 'Granulocytes',  
 '8': 'Erythroids',  
 '22': 'Megakaryocytes',  
 '16': 'Erythroids',  
 '1': 'GMP',  
 '17': 'Megakaryocytes',  
 '4': 'Granulocytes',  
 '19': 'Erythroids',  
 '9': 'Erythroids',  
 '6': 'Monocytes',  
 '20': 'Lymphoid',  
 '14': 'Erythroids',  
 '15': 'Erythroids',  
 '21': 'DC'}

[23]:

adata.obs["cell\_type"] = [annotation\_rev[i] for i in adata.obs.louvain]

[24]:

# check results  
sc.pl.draw\_graph(adata, color=['cell\_type'],  
 legend\_loc='on data')

... storing 'cell\_type' as categorical

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_34_1.png>

###### 8.2. Make annotation (2)[¶](#8.2.-Make-annotation-(2))

We’ll make another annotation manually for each louvain clusters

[25]:

sc.pl.draw\_graph(adata, color=['louvain', 'cell\_type'],  
 legend\_loc='on data')

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_36_0.png>

[26]:

annotation\_2 = {'5': 'MEP\_0',  
 '15': 'Ery\_0',  
 '10': 'Ery\_1',  
 '16': 'Ery\_2',  
 '14': 'Ery\_3',  
 '9': 'Ery\_4',  
 '8': 'Ery\_5',  
 '19': 'Ery\_6',  
 '3': 'Ery\_7',  
 '12': 'Ery\_8',  
 '18': 'Ery\_9',  
 '17': 'Mk\_0',  
 '22': 'Mk\_0',  
 '11': 'GMP\_0',  
 '1': 'GMP\_1',  
 '0': 'GMPl\_0',  
 '7': 'Gran\_0',  
 '13': 'Gran\_1',  
 '4': 'Gran\_2',  
 '6': 'Mo\_0',  
 '2': 'Mo\_1',  
 '21': 'DC\_0',  
 '20': 'Lym\_0'}

[27]:

adata.obs["louvain\_annot"] = [annotation\_2[i] for i in adata.obs.louvain]

[28]:

# check result  
sc.pl.draw\_graph(adata, color=['louvain\_annot', 'cell\_type'],  
 legend\_loc='on data')

... storing 'louvain\_annot' as categorical

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_39_1.png>

We’ve done secveral scRNA-preprocessing steps; filtering, normalization, clustering, dimensional reduction. In the next step, we’ll do GRN inference, network analysis, and in silico simulation based on these information.

##### 9. (Option) Select cell cluster[¶](#9.-(Option)-Select-cell-cluster)

In this tutorial, we are using scRNA-seq data of hematopoiesis. In the latter part, we will focus on the cell fate decision in the myeloid lineage. So we remove non-myeloid cell cluster, DC and Lymphoid now.

[29]:

adata.obs.cell\_type.unique()

[29]:

[MEP, Monocytes, Erythroids, Granulocytes, late\_GMP, GMP, Megakaryocytes, Lymphoid, DC]  
Categories (9, object): [MEP, Monocytes, Erythroids, Granulocytes, ..., GMP, Megakaryocytes, Lymphoid, DC]

[30]:

cell\_of\_interest = adata.obs.index[~adata.obs.cell\_type.isin(["Lymphoid", "DC"])]  
adata = adata[cell\_of\_interest, :]

[31]:

# check result  
sc.pl.draw\_graph(adata, color=['louvain\_annot', 'cell\_type'],  
 legend\_loc='on data')

<_images/notebooks_03_scRNA-seq_data_preprocessing_scanpy_preprocessing_with_Paul_etal_2015_data_44_0.png>

##### 9. Save data[¶](#9.-Save-data)

[32]:

adata.write\_h5ad("data/Paul\_etal\_15.h5ad")

[ ]:

#### B. scRNA-seq data preprocessing with Seurat[¶](#b-scrna-seq-data-preprocessing-with-seurat)

R notebook … comming soon

### 4. Network analysis[¶](#network-analysis)

celloracle reconstructs cluster-specicif GRNs using scRNA-seq dataset and TF binding information which was made in the previous steps. Then several network analysis methods will be applied to the cluster specific GRNs. This step includes network structure analysis, network score analysis, and key regulatory gene identification.

Python notebook

#### 0. Import libraries[¶](#0.-Import-libraries)

[1]:

# 0. Import  
  
import os  
import sys  
  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import scanpy as sc  
import seaborn as sns  
from tqdm import tqdm\_notebook as tqdm

[2]:

import celloracle as co

[84]:

# visualization settings  
%config InlineBackend.figure\_format = 'retina'  
%matplotlib inline  
  
plt.rcParams["figure.figsize"] = [9, 6]  
plt.rcParams["savefig.dpi"] = 600

##### 0.1. Check installation[¶](#0.1.-Check-installation)

Celloracle use some R libraries in network analysis. Check installation.

[4]:

co.network\_analysis.test\_R\_libraries\_installation()

checking R library installation: igraph -> OK  
checking R library installation: linkcomm -> OK  
checking R library installation: rnetcarto -> OK

##### 0.2. Make a folder to save graph[¶](#0.2.-Make-a-folder-to-save-graph)

[5]:

save\_folder = "figures"  
os.makedirs(save\_folder, exist\_ok=True)

#### 1. Load data[¶](#1.-Load-data)

##### 1.1. Load processed gene expression data (anndata)[¶](#1.1.-Load-processed-gene-expression-data-(anndata))

Please refer to the previous notebook in the tutorial for an example of how to process scRNA-seq data.

[12]:

# load data. !!Replace the data path below when you use another data.  
adata = sc.read\_h5ad("../03\_scRNA-seq\_data\_preprocessing/data/Paul\_etal\_15.h5ad")

##### 1.2. Load TF data.[¶](#1.2.-Load-TF-data.)

celloracle need TF information, which contains lists of regulatory candidata gene. There are several way to make such TF information. TF information can be made from scATAC-seq data or Bulk ATAC-seq data. Please refer to the first step of the tutorial.

If you do not have your scATAC-seq data, you can use some builtin data in celloracle. For example, celloracle have a TFinfo which was made with various kind of tissue/cell-types in mouse ATAC-seq atlas dataset (<http://atlas.gs.washington.edu/mouse-atac/>).

This TFinfo is ready. You can load and use it with following command.

[13]:

# Load TF info which was made from mouse cell atlas dataset.  
TFinfo\_df = co.data.load\_TFinfo\_df\_mm9\_mouse\_atac\_atlas()  
  
# check data  
TFinfo\_df.head()

[13]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | peak\_id | gene\_short\_name | 9430076c15rik | Ac002126.6 | Ac012531.1 | Ac226150.2 | Afp | Ahr | Ahrr | Aire | ... | Znf784 | Znf8 | Znf816 | Znf85 | Zscan10 | Zscan16 | Zscan22 | Zscan26 | Zscan31 | Zscan4 |
| 0 | chr10\_100050979\_100052296 | 4930430F08Rik | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | chr10\_101006922\_101007748 | SNORA17 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 2 | chr10\_101144061\_101145000 | Mgat4c | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| 3 | chr10\_10148873\_10149183 | 9130014G24Rik | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | chr10\_10149425\_10149815 | 9130014G24Rik | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

5 rows × 1095 columns

#### 2. Initiate Oracle object[¶](#2.-Initiate-Oracle-object)

Basically all information will be stored and proccessed inside Oracle object. Processed gene expression data (anndata) and TFinfo are necessary to initiate Oracle object.

[14]:

# make Oracle object  
oracle = co.Oracle()

##### 2.1. load gene expression data into oracle object.[¶](#2.1.-load-gene-expression-data-into-oracle-object.)

[15]:

# The anndata shoud include (1) gene expression count, (2) clustering information, (3) trajectory (dimensional reduction embeddings) data.  
# Please refer to another notebook for the detail of anndata preprocessing.  
  
# In this notebook, we use raw mRNA count as an input of Oracle object.  
adata.X = adata.raw.X.copy()  
  
# Instantiate Oracle object. All calculation will be done in this object.  
oracle.import\_anndata\_as\_raw\_count(adata=adata,  
 cluster\_column\_name="louvain\_annot",  
 embedding\_name="X\_draw\_graph\_fa",  
 transform="natural\_log")

##### 2.2. load TFinfo into oracle object[¶](#2.2.-load-TFinfo-into-oracle-object)

[16]:

# If you can load TF info dataframe with the following code.  
oracle.import\_TF\_data(TF\_info\_matrix=TFinfo\_df)  
  
# Instead of the way avove, you can use TF "dictionary" with the following code.  
# oracle.import\_TF\_data(TFdict=TFinfo\_dictionary)

##### 2.3. (Optional) Add TF info manually[¶](#2.3.-(Optional)-Add-TF-info-manually)

We mainly use TF info data made from scATAC-seq data. Besides we can also add any information about TF-target gene pair manually.

For example if there is a study or database about TF-target pair, you can use such information by the following way.

###### 2.3.1. Make TF info dictionary manually[¶](#2.3.1.-Make-TF-info-dictionary-manually)

Here, we introduce most simple way to add TF binding information using TF binding data in the supplemental table 4 in (<http://doi.org/10.1016/j.cell.2015.11.013>).

[17]:

# We have TF and its target gene information. This is from a supplemental Fig of Paul et. al, (2015).  
Paul\_15\_data = pd.read\_csv("TF\_data\_in\_Paul15.csv")  
Paul\_15\_data

[17]:

|  |  |  |
| --- | --- | --- |
|  | TF | Target\_genes |
| 0 | Cebpa | Abcb1b, Acot1, C3, Cnpy3, Dhrs7, Dtx4, Edem2, ... |
| 1 | Irf8 | Abcd1, Aif1, BC017643, Cbl, Ccdc109b, Ccl6, d6... |
| 2 | Irf8 | 1100001G20Rik, 4732418C07Rik, 9230105E10Rik, A... |
| 3 | Klf1 | 2010011I20Rik, 5730469M10Rik, Acsl6, Add2, Ank... |
| 4 | Sfpi1 | 0910001L09Rik, 2310014H01Rik, 4632428N05Rik, A... |

[18]:

# Make dictionary: Key is TF, Value is list of Target gene  
TF\_to\_TG\_dictionary = {}  
  
for TF, TGs in zip(Paul\_15\_data.TF, Paul\_15\_data.Target\_genes):  
 # convert target gene to list  
 TG\_list = TGs.replace(" ", "").split(",")  
 # store target gene list in a dictionary  
 TF\_to\_TG\_dictionary[TF] = TG\_list  
  
# We have to make a dictionary, in which a Key is Target gene and value is TF.  
# We inverse the dictionary above using a utility function in celloracle.  
TG\_to\_TF\_dictionary = co.utility.inverse\_dictionary(TF\_to\_TG\_dictionary)

###### 2.3.2. Add TF informatio dictionary into the oracle object[¶](#2.3.2.-Add-TF-informatio-dictionary-into-the-oracle-object)

[19]:

# Add TF information  
oracle.addTFinfo\_dictionary(TG\_to\_TF\_dictionary)

#### 3. Knn imputation[¶](#3.-Knn-imputation)

Celloracle uses almost the same strategy as velocyto for visualization of trajectory, which requires knn imputation to improves the accuracy of cell transition calculation. Besides, PCA and PC selection are required for knn imputation.

##### 3.1. PCA[¶](#3.1.-PCA)

[85]:

# perform PCA  
oracle.perform\_PCA()  
  
# select important PCs  
plt.plot(np.cumsum(oracle.pca.explained\_variance\_ratio\_)[:100])  
n\_comps = np.where(np.diff(np.diff(np.cumsum(oracle.pca.explained\_variance\_ratio\_))>0.002))[0][0]  
plt.axvline(n\_comps, c="k")  
print(n\_comps)  
n\_comps = min(n\_comps, 50)

45

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_25_1.png>

##### 3.2. Knn imputation[¶](#3.2.-Knn-imputation)

[21]:

n\_cell = oracle.adata.shape[0]  
print(f"cell number is :{n\_cell}")

cell number is :2671

[22]:

k = int(0.025\*n\_cell)  
print(f"Auto-selected k is :{k}")

Auto-selected k is :66

[23]:

oracle.knn\_imputation(n\_pca\_dims=n\_comps, k=k, balanced=True, b\_sight=k\*8,  
 b\_maxl=k\*4, n\_jobs=4)

#### 4. Save and Load.[¶](#4.-Save-and-Load.)

Celloracle has some custom-class; Links, Oracle, TFinfo, etc., You can save such object using “to\_hdf5”. The file will be saved as a hdf5 file.

To load saved file, pleasae use “load\_hdf5” function.

[24]:

# save oracle object.  
oracle.to\_hdf5("Paul\_15\_data.celloracle.oracle")

[68]:

# load file.  
# oracle = co.load\_hdf5("Paul\_15\_data.celloracle.oracle")

#### 4. GRN calculation[¶](#4.-GRN-calculation)

Next step is constructing a cluster-specific GRN for all clusters.

GRN can be calculated at any arbitrary unit.

With “get\_links” function, you can calculate cluster-specific GRNs at any unit, and the function returns GRNs as Links object. Links object stores inferred GRNs and its metadata. You can do network analysis with Links object.

In the example below, we will make GRNs based on “louvain\_annot” clustering unit.

You can use any clustering unit for GRN calculation as long as clustering information is saved in anndata.

[86]:

# check data  
sc.pl.draw\_graph(oracle.adata, color="louvain\_annot")

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_35_0.png>

[28]:

# calculate GRN for each population in "louvain\_annot" clustering unit.  
# This step may take long time.  
links = oracle.get\_links(cluster\_name\_for\_GRN\_unit="louvain\_annot", alpha=10,  
 verbose\_level=2, test\_mode=False)

inferring GRN for Ery\_0...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_1...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_2...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_3...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_4...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_5...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_6...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_7...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_8...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Ery\_9...  
method: bagging\_ridge  
alpha: 10

inferring GRN for GMP\_0...  
method: bagging\_ridge  
alpha: 10

inferring GRN for GMP\_1...  
method: bagging\_ridge  
alpha: 10

inferring GRN for GMPl\_0...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Gran\_0...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Gran\_1...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Gran\_2...  
method: bagging\_ridge  
alpha: 10

inferring GRN for MEP\_0...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Mk\_0...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Mo\_0...  
method: bagging\_ridge  
alpha: 10

inferring GRN for Mo\_1...  
method: bagging\_ridge  
alpha: 10

Links object have some metadata.

[33]:

links.palette

[33]:

|  |  |
| --- | --- |
|  | palette |
| Ery\_0 | #7D87B9 |
| Ery\_1 | #BEC1D4 |
| Ery\_2 | #D6BCC0 |
| Ery\_3 | #BB7784 |
| Ery\_4 | #8E063B |
| Ery\_5 | #4A6FE3 |
| Ery\_6 | #8595E1 |
| Ery\_7 | #B5BBE3 |
| Ery\_8 | #E6AFB9 |
| Ery\_9 | #E07B91 |
| GMP\_0 | #D33F6A |
| GMP\_1 | #11C638 |
| GMPl\_0 | #8DD593 |
| Gran\_0 | #C6DEC7 |
| Gran\_1 | #EAD3C6 |
| Gran\_2 | #F0B98D |
| MEP\_0 | #0FCFC0 |
| Mk\_0 | #9CDED6 |
| Mo\_0 | #D5EAE7 |
| Mo\_1 | #F3E1EB |

For the visualization below, we change the order of clusters.

[35]:

# For the visualization below, we define and save order of clusters.  
order = ['MEP\_0', 'Mk\_0','Ery\_0', 'Ery\_1', 'Ery\_2', 'Ery\_3', 'Ery\_4', 'Ery\_5',  
 'Ery\_6', 'Ery\_7', 'Ery\_8', 'Ery\_9','GMP\_0', 'GMP\_1',  
 'GMPl\_0', 'Mo\_0', 'Mo\_1', 'Gran\_0', 'Gran\_1', 'Gran\_2']  
links\_louvain.palette = links\_louvain.palette.loc[order]  
links\_louvain.palette

[35]:

|  |  |
| --- | --- |
|  | palette |
| MEP\_0 | #0FCFC0 |
| Mk\_0 | #9CDED6 |
| Ery\_0 | #7D87B9 |
| Ery\_1 | #BEC1D4 |
| Ery\_2 | #D6BCC0 |
| Ery\_3 | #BB7784 |
| Ery\_4 | #8E063B |
| Ery\_5 | #4A6FE3 |
| Ery\_6 | #8595E1 |
| Ery\_7 | #B5BBE3 |
| Ery\_8 | #E6AFB9 |
| Ery\_9 | #E07B91 |
| GMP\_0 | #D33F6A |
| GMP\_1 | #11C638 |
| GMPl\_0 | #8DD593 |
| Mo\_0 | #D5EAE7 |
| Mo\_1 | #F3E1EB |
| Gran\_0 | #C6DEC7 |
| Gran\_1 | #EAD3C6 |
| Gran\_2 | #F0B98D |

#### 5. Network preprocessing[¶](#5.-Network-preprocessing)

##### 5.1. Filter network edges[¶](#5.1.-Filter-network-edges)

Celloracle leverages bagging ridge or Bayesian ridge regression for network inference. These methods provide a network edge strength as a distribution rather than point value. We can use the distribution of the network edge strength to know the certainness of the connection.

We will do filtering of network edge before starting network analysis.

1. Remove uncertain network edge based on the p-value.
2. Remove weak network edge. In this tutorial, we pick up top 2000 edges in terms of network strength.

The raw network data is stored as an attribute, “links\_dict,” while filtered network data will be stored in “filtered\_links.” So the raw network data will not be erased after the filtering. You can come back to the filtering process and do the filtering again if you do not like filtering results.

[32]:

links.filter\_links(p=0.001, weight="coef\_abs", thread\_number=2000)

##### 5.2. Degree distribution[¶](#5.2.-Degree-distribution)

Network degree, which means the number of edges for each gene, is one of the important metrics to know the network analysis. The degree distribution may change depending on the filtering threshold.

[87]:

links.plot\_degree\_distributions(plot\_model=True, save=f"{save\_folder}/degree\_distribution/")

Ery\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_1.png>

Ery\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_3.png>

Ery\_2

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_5.png>

Ery\_3

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_7.png>

Ery\_4

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_9.png>

Ery\_5

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_11.png>

Ery\_6

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_13.png>

Ery\_7

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_15.png>

Ery\_8

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_17.png>

Ery\_9

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_19.png>

GMP\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_21.png>

GMP\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_23.png>

GMPl\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_25.png>

Gran\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_27.png>

Gran\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_29.png>

Gran\_2

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_31.png>

MEP\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_33.png>

Mk\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_35.png>

Mo\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_37.png>

Mo\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_44_39.png>

##### 5.3. Calculate netowrk score[¶](#5.3.-Calculate-netowrk-score)

Next, we calculate several network score using some R libraries. Please make sure that R libraries are installed in your PC before running the command below.

[39]:

# calculate network scores. It takes several minuts.  
links.get\_score()

processing... batch 1/5  
Ery\_0: finished.  
Ery\_1: finished.  
Ery\_2: finished.  
Ery\_3: finished.  
processing... batch 2/5  
Ery\_4: finished.  
Ery\_5: finished.  
Ery\_6: finished.  
Ery\_7: finished.  
processing... batch 3/5  
Ery\_8: finished.  
Ery\_9: finished.  
GMP\_0: finished.  
GMP\_1: finished.  
processing... batch 4/5  
GMPl\_0: finished.  
Gran\_0: finished.  
Gran\_1: finished.  
Gran\_2: finished.  
processing... batch 5/5  
MEP\_0: finished.  
Mk\_0: finished.  
Mo\_0: finished.  
Mo\_1: finished.  
the scores are saved in ./louvain\_annot/

The score is stored as a attribute “merged\_score”, and score will also be saved in a folder.

[57]:

links.merged\_score.head()

[57]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | degree\_all | degree\_in | degree\_out | clustering\_coefficient | clustering\_coefficient\_weighted | degree\_centrality\_all | degree\_centrality\_in | degree\_centrality\_out | betweenness\_centrality | closeness\_centrality | ... | assortative\_coefficient | average\_path\_length | community\_edge\_betweenness | community\_random\_walk | community\_eigenvector | module | connectivity | participation | role | cluster |
| Stat3 | 91 | 0 | 91 | 0.019780 | 0.020122 | 0.167279 | 0.000000 | 0.167279 | 0 | 0.000013 | ... | -0.172207 | 2.475244 | 1 | 1 | 1 | 0 | 4.052023 | 0.632291 | Connector Hub | Ery\_0 |
| Mycn | 28 | 0 | 28 | 0.002646 | 0.001828 | 0.051471 | 0.000000 | 0.051471 | 0 | 0.000004 | ... | -0.172207 | 2.475244 | 2 | 1 | 5 | 2 | 2.347146 | 0.625000 | Connector | Ery\_0 |
| Zbtb1 | 27 | 0 | 27 | 0.005698 | 0.008576 | 0.049632 | 0.000000 | 0.049632 | 0 | 0.000011 | ... | -0.172207 | 2.475244 | 3 | 1 | 5 | 1 | 2.275431 | 0.666667 | Connector | Ery\_0 |
| E2f4 | 186 | 3 | 183 | 0.009474 | 0.011623 | 0.341912 | 0.005515 | 0.336397 | 2788 | 0.000010 | ... | -0.172207 | 2.475244 | 4 | 1 | 3 | 0 | 8.496354 | 0.637760 | Connector Hub | Ery\_0 |
| Ybx1 | 71 | 9 | 62 | 0.027364 | 0.027485 | 0.130515 | 0.016544 | 0.113971 | 1153 | 0.000004 | ... | -0.172207 | 2.475244 | 5 | 11 | 4 | 3 | 5.433857 | 0.564967 | Connector Hub | Ery\_0 |

5 rows × 22 columns

##### 5.4. Save[¶](#5.4.-Save)

Save processed GRN. The Links file will also be used in the next notebook; “in silico simulation with GRNs”.

[42]:

# Save Links object.  
links.to\_hdf5(file\_path="links.celloracle.links")  
  
# You can load files with the following command.  
#links = co.load\_hdf5(file\_path="links.celloracle.links")

If you are not interested in the network score analysis and just want to do simulation, you can skip the analysis below and go to the next notebook.

#### 6. Network analysis; Network score for each gene[¶](#6.-Network-analysis;-Network-score-for-each-gene)

Links class has many functions to visualize network score. See the documentation for the details of the functions.

##### 6.1. Network score in each cluster[¶](#6.1.-Network-score-in-each-cluster)

[44]:

# check cluster  
links.cluster

[44]:

['Ery\_0',  
 'Ery\_1',  
 'Ery\_2',  
 'Ery\_3',  
 'Ery\_4',  
 'Ery\_5',  
 'Ery\_6',  
 'Ery\_7',  
 'Ery\_8',  
 'Ery\_9',  
 'GMP\_0',  
 'GMP\_1',  
 'GMPl\_0',  
 'Gran\_0',  
 'Gran\_1',  
 'Gran\_2',  
 'MEP\_0',  
 'Mk\_0',  
 'Mo\_0',  
 'Mo\_1']

[88]:

# Visualize top n-th genes that have high scores.  
links.plot\_scores\_as\_rank(cluster="MEP\_0", n\_gene=30, save=f"{save\_folder}/ranked\_score")

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_54_0.png>

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_54_1.png>

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_54_2.png>

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_54_3.png>

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_54_4.png>

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_54_5.png>

##### 6.2. Network score comparison between two clusters[¶](#6.2.-Network-score-comparison-between-two-clusters)

By comparing network scores between two clusters, we can analyze the difference of GRN structure.

[89]:

plt.ticklabel\_format(style='sci',axis='y',scilimits=(0,0))  
links.plot\_score\_comparison\_2D(value="eigenvector\_centrality",  
 cluster1="MEP\_0", cluster2="GMPl\_0",  
 percentile=98, save=f"{save\_folder}/score\_comparison")

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_57_0.png>

[90]:

plt.ticklabel\_format(style='sci',axis='y',scilimits=(0,0))  
links.plot\_score\_comparison\_2D(value="betweenness\_centrality",  
 cluster1="MEP\_0", cluster2="GMPl\_0",  
 percentile=98, save=f"{save\_folder}/score\_comparison")

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_58_0.png>

[91]:

plt.ticklabel\_format(style='sci',axis='y',scilimits=(0,0))  
links.plot\_score\_comparison\_2D(value="degree\_centrality\_all",  
 cluster1="MEP\_0", cluster2="GMPl\_0",  
 percentile=98, save=f"{save\_folder}/score\_comparison")

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_59_0.png>

##### 6.3. Network score dynamics[¶](#6.3.-Network-score-dynamics)

We can analyze how gene function/importance change over the differentiation by the following analysis.

Gata2 is known to play an important role in the early progenitor population; MEP, GMP. The following results can recapitulate the Gata2 feature.

[92]:

# Visualize Gata2 network score dynamics  
links.plot\_score\_per\_cluster(goi="Gata2", save=f"{save\_folder}/network\_score\_per\_gene/")

Gata2

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_61_1.png>

##### 6.4. Gene cartography analysis[¶](#6.4.-Gene-cartography-analysis)

Gene cartography is a method for gene network analysis. The method classifies the gene into several groups using the network module structure and gene connections. It provides us an insight about the regulatory mechanism per each gene. Please read the paper for the details of gene cartography (<https://www.nature.com/articles/nature03288>)

The calculations of gene cartography will be performed for the GRN in each cluster. Thus we can know how gene role change by comparing the cartography in each cluster.

[93]:

# plot cartography as a scatter plot  
links.plot\_cartography\_scatter\_per\_cluster(scatter=True,  
 kde=False,  
 gois=["Gata1", "Gata2", "Sfpi1"],  
 auto\_gene\_annot=False,  
 percentile=99.5,  
 args\_dot={"n\_levels": 105},  
 args\_line={"c":"gray"}, save=f"{save\_folder}/cartography")

Ery\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_1.png>

Ery\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_3.png>

Ery\_2

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_5.png>

Ery\_3

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_7.png>

Ery\_4

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_9.png>

Ery\_5

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_11.png>

Ery\_6

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_13.png>

Ery\_7

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_15.png>

Ery\_8

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_17.png>

Ery\_9

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_19.png>

GMP\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_21.png>

GMP\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_23.png>

GMPl\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_25.png>

Gran\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_27.png>

Gran\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_29.png>

Gran\_2

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_31.png>

MEP\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_33.png>

Mk\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_35.png>

Mo\_0

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_37.png>

Mo\_1

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_63_39.png>

[94]:

# plot the results of cartography analysis  
plt.figure(figsize=[3,6])  
links.plot\_cartography\_term(goi="Gata2", save=f"{save\_folder}/cartography")

Gata2

/home/k/anaconda3/envs/test/lib/python3.6/site-packages/pandas/core/indexing.py:1418: FutureWarning:  
Passing list-likes to .loc or [] with any missing label will raise  
KeyError in the future, you can use .reindex() as an alternative.  
  
See the documentation here:  
https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#deprecate-loc-reindex-listlike  
 return self.\_getitem\_tuple(key)

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_64_2.png>

#### 7. Network analysis; Network score distribution[¶](#7.-Network-analysis;-Network-score-distribution)

Next, we visualize the distribution of network score to get insight into the global trend about the structure of GRNs.

##### 7.1. Distribution of network degree[¶](#7.1.-Distribution-of-network-degree)

[95]:

plt.subplots\_adjust(left=0.15, bottom=0.3)  
plt.ylim([0,0.040])  
links.plot\_score\_discributions(values=["degree\_centrality\_all"], method="boxplot", save=f"{save\_folder}")

degree\_centrality\_all

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_67_1.png>

[96]:

plt.subplots\_adjust(left=0.15, bottom=0.3)  
plt.ylim([0, 0.40])  
links.plot\_score\_discributions(values=["eigenvector\_centrality"], method="boxplot", save=f"{save\_folder}")

eigenvector\_centrality

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_68_1.png>

##### 7.2. Distribution of netowrk entolopy[¶](#7.2.-Distribution-of-netowrk-entolopy)

[97]:

plt.subplots\_adjust(left=0.15, bottom=0.3)  
links.plot\_network\_entropy\_distributions(save=f"{save\_folder}")

<_images/notebooks_04_Network_analysis_Network_analysis_with_with_Paul_etal_2015_data_70_0.png>

In the example below, we did several network analysis and showed that several TFs have high network scores. The TFs, for example “Gata2”, “Gata1”, “Klf1”, “E2f1”, are known to play an important role in MEP. Using the network score, we may pick up cluster-specific key TFs.

In this step, however, we cannot know the specific function or relationship between cell fate. We just pick up several candidate genes for Key TFs here, and in the next analysis, we investigate their function and relationship between cell fate by the simulation with GRNs.

### 5. Simulation with GRNs[¶](#simulation-with-grns)

celloracle can simulate an affect of gene perturbation using GRNs. This analysis provides insight into the regulatory mechanism of cel state from a viewpoint of TF and GRN.

Python notebook

#### 0. Import libraries[¶](#0.-Import-libraries)

##### 0.1. Import public libraries[¶](#0.1.-Import-public-libraries)

[1]:

import os  
import sys  
  
import matplotlib.colors as colors  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import scanpy as sc  
import seaborn as sns

[3]:

import celloracle as co

[15]:

#plt.rcParams["font.family"] = "arial"  
plt.rcParams["figure.figsize"] = [9,6]  
%config InlineBackend.figure\_format = 'retina'  
plt.rcParams["savefig.dpi"] = 600  
  
%matplotlib inline

##### 0.1. Make a folder to save graph[¶](#0.1.-Make-a-folder-to-save-graph)

[5]:

# make folder to save plots  
save\_folder = "figures"  
os.makedirs(save\_folder, exist\_ok=True)

#### 1. Load data[¶](#1.-Load-data)

##### 1.1. Load processed oracle object[¶](#1.1.-Load-processed-oracle-object)

Load the oracle object. See the previous notebook for the detil of how to prepare oracle object.

[7]:

oracle = co.load\_hdf5("../04\_Network\_analysis/Paul\_15\_data.celloracle.oracle")

##### 1.2. Load inferred GRNs[¶](#1.2.-Load-inferred-GRNs)

In the previous notebook, we already calculated GRNs. We use this GRNs for simulation.

For the simulation with GRNs, we import GRNs which was saved in Links object.

[8]:

links = co.load\_hdf5("../04\_Network\_analysis/links.celloracle.links")

#### 3. Make predictive models for simulation[¶](#3.-Make-predictive-models-for-simulation)

We will construct regression models again by using the filtered GRN structure. This process take less time than the GRN inference in the previous notebook because we use only significant TFs to predict target gene rather than all regulatory candidate TFs.

[12]:

links.filter\_links()  
oracle.get\_cluster\_specific\_TFdict\_from\_Links(links\_object=links)  
oracle.fit\_GRN\_for\_simulation(alpha=10, use\_cluster\_specific\_TFdict=True)

calculating GRN using cluster specicif TF dict...  
calculating GRN in Ery\_0

genes\_in\_gem: 1999  
models made for 1074 genes  
calculating GRN in Ery\_1

genes\_in\_gem: 1999  
models made for 1092 genes  
calculating GRN in Ery\_2

genes\_in\_gem: 1999  
models made for 1064 genes  
calculating GRN in Ery\_3

genes\_in\_gem: 1999  
models made for 1105 genes  
calculating GRN in Ery\_4

genes\_in\_gem: 1999  
models made for 1102 genes  
calculating GRN in Ery\_5

genes\_in\_gem: 1999  
models made for 1116 genes  
calculating GRN in Ery\_6

genes\_in\_gem: 1999  
models made for 1097 genes  
calculating GRN in Ery\_7

genes\_in\_gem: 1999  
models made for 1062 genes  
calculating GRN in Ery\_8

genes\_in\_gem: 1999  
models made for 1117 genes  
calculating GRN in Ery\_9

genes\_in\_gem: 1999  
models made for 1121 genes  
calculating GRN in GMP\_0

genes\_in\_gem: 1999  
models made for 1107 genes  
calculating GRN in GMP\_1

genes\_in\_gem: 1999  
models made for 1104 genes  
calculating GRN in GMPl\_0

genes\_in\_gem: 1999  
models made for 1089 genes  
calculating GRN in Gran\_0

genes\_in\_gem: 1999  
models made for 1067 genes  
calculating GRN in Gran\_1

genes\_in\_gem: 1999  
models made for 1076 genes  
calculating GRN in Gran\_2

genes\_in\_gem: 1999  
models made for 1105 genes  
calculating GRN in MEP\_0

genes\_in\_gem: 1999  
models made for 1152 genes  
calculating GRN in Mk\_0

genes\_in\_gem: 1999  
models made for 1114 genes  
calculating GRN in Mo\_0

genes\_in\_gem: 1999  
models made for 1085 genes  
calculating GRN in Mo\_1

genes\_in\_gem: 1999  
models made for 1074 genes

#### 4. in silico Perturbation-simulation[¶](#4.-in-silico-Perturbation-simulation)

Next, we will do simulation. Using signal progagation algorithm of celloracle, we can predict what happens if we perturb TF(s).

See the celloracle paper for the details.

In this notebook, we’ll show an example of how to do in silico perturb simulation; we’ll simulate Gata1 KO in the hematopiesis. Previous studiy have shown that Gata1 is one of the important TFs that regulate cell fate decision in the myeloid progenitors and Gata1 have positive effect for the erythroid cell differentiation.

##### 4.1. Check gene expression pattern.[¶](#4.1.-Check-gene-expression-pattern.)

[26]:

# check gene expression  
goi = "Gata1"  
sc.pl.draw\_graph(oracle.adata, color=[goi, oracle.cluster\_column\_name], layer="imputed\_count", use\_raw=False, cmap="viridis")

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_15_0.png>

[33]:

# plot gene expression in histogram  
sc.get.obs\_df(oracle.adata, keys=[goi], layer="imputed\_count").hist()  
plt.show()

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_16_0.png>

##### 4.1. calculate future gene expression after perturbation.[¶](#4.1.-calculate-future-gene-expression-after-perturbation.)

Although you can set any arbitrary gene expression value for perturb condition, we recommend avoiding too high value if you simulate overexpression. In the example below, if you set Gata1 gene expression to 100, for example, it may lead to biologically infeasible results.

Here we will simulate Gata1 KO; we will predict what happens to the cells if Gata1 gene expression changed into 0.

[34]:

# Enter perturbation conditions and simulate signal propagation after the perturbation.  
oracle.simulate\_shift(perturb\_condition={goi: 0.0},  
 n\_propagation=3)

##### 4.2. calculate transition probability between cells[¶](#4.2.-calculate-transition-probability-between-cells)

In the step avobe, we simulated what happens if we perturb TF(s). This means we simulated future gene expression change after perturbation. This prediction is based on itelative calculations of signal propagations inside the GRN. Here we performed just small number of the calculation, thus the we simulated short future.

Next step, we will calculate the propability of cell state transition based on the simulated data. Using this cell state transition probability between cells, we can predict how cell state changes after perturbation, enabling the analysis of the relationship between genes and cell state determination.

This transition probability will be used in two ways.

1. Visualization of directed trjectory graph.
2. Markof simulation.

[35]:

oracle.estimate\_transition\_prob(n\_neighbors=200, knn\_random=True, sampled\_fraction=0.5)  
oracle.calculate\_embedding\_shift(sigma\_corr = 0.05)  
oracle.calculate\_grid\_arrows(smooth=0.8, steps=(40, 40), n\_neighbors=300)

/home/k/anaconda3/envs/test/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3326: FutureWarning: arrays to stack must be passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such as generators is deprecated as of NumPy 1.16 and will raise an error in the future.  
 exec(code\_obj, self.user\_global\_ns, self.user\_ns)  
WARNING:root:Nans encountered in corrcoef and corrected to 1s. If not identical cells were present it is probably a small isolated cluster converging after imputation.

##### 4.3. Visualization[¶](#4.3.-Visualization)

###### 4.3.1. Detailed directed trajectory graph[¶](#4.3.1.-Detailed-directed-trajectory-graph)

[36]:

plt.figure(None,(6,6))  
quiver\_scale = 40  
  
  
ix\_choice = np.random.choice(oracle.adata.shape[0], size=int(oracle.adata.shape[0]/1.), replace=False)  
  
embedding = oracle.adata.obsm[oracle.embedding\_name]  
  
plt.scatter(embedding[ix\_choice, 0], embedding[ix\_choice, 1],  
 c="0.8", alpha=0.2, s=38, edgecolor=(0,0,0,1), lw=0.3, rasterized=True)  
  
quiver\_kwargs=dict(headaxislength=7, headlength=11, headwidth=8,  
 linewidths=0.35, width=0.0045,edgecolors="k",  
 color=oracle.colorandum[ix\_choice], alpha=1)  
plt.quiver(embedding[ix\_choice, 0], embedding[ix\_choice, 1],  
 oracle.delta\_embedding[ix\_choice, 0], oracle.delta\_embedding[ix\_choice, 1],  
 scale=quiver\_scale, \*\*quiver\_kwargs)  
  
plt.axis("off")  
#plt.savefig(f"{save\_folder}/full\_arrows{goi}.png", transparent=True)

[36]:

(-10815.27020913708, 10950.84121716522, -10711.36365432337, 10949.477199695968)

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_22_1.png>

###### 4.3.2. Grid graph[¶](#4.3.2.-Grid-graph)

[37]:

# plot whole graph  
plt.figure(None,(10,10))  
oracle.plot\_grid\_arrows(quiver\_scale=2.0,  
 scatter\_kwargs\_dict={"alpha":0.35, "lw":0.35,  
 "edgecolor":"0.4", "s":38,  
 "rasterized":True},  
 min\_mass=0.015, angles='xy', scale\_units='xy',  
 headaxislength=2.75,  
 headlength=5, headwidth=4.8, minlength=1.5,  
 plot\_random=False, scale\_type="relative")  
#plt.savefig(f"{save\_folder}/vectorfield\_{goi}.png", transparent=True)

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_24_0.png>

##### 4.4. Markov simulation to analyze the effects of perturbation on cell fate transition[¶](#4.4.-Markov-simulation-to-analyze-the-effects-of-perturbation-on-cell-fate-transition)

Based on the transition probability, markov simulation will be performed to simulate cell state transition.

###### 4.4.1. Do simulation[¶](#4.4.1.-Do-simulation)

We simulate with a parameter, “n\_steps=200” and in the following example. It means that

1. we will do 200 times of iterative simulations to predict how cell changes over time
2. and we will do 5 round of simulations for each cells.

[83]:

%%time  
# n\_steps is the number of steps in markov simulation.  
# n\_duplication is the number of technical duplicate for the simulation  
oracle.run\_markov\_chain\_simulation(n\_steps=200, n\_duplication=5)

CPU times: user 1.33 s, sys: 0 ns, total: 1.33 s  
Wall time: 1.33 s

###### 4.4.2. Check the results of simulation focusing on a specific cells[¶](#4.4.2.-Check-the-results-of-simulation-focusing-on-a-specific-cells)

Check the results of simulation by picking some cells and visualize their transition trajectory.

[88]:

#np.random.seed(12)  
# randomly pick up 3 cells  
cells = oracle.adata.obs.index.values[np.random.choice(oracle.ixs\_mcmc, 3)]  
  
# visualize the simulated results of cell transition after perturbation  
for k in cells:  
 print(f"cell {k}")  
 plt.figure(figsize=[9, 3])  
 for j, i in enumerate([0, 20, 50]): # time points  
 plt.subplot(1, 3, (j+1))  
 oracle.plot\_mc\_result\_as\_trajectory(k, range(0, i))  
 plt.title(f"simulation step: 0~{i}")  
 plt.axis("off")  
 plt.show()

cell 1961

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_32_1.png>

cell 43

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_32_3.png>

cell 1567

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_32_5.png>

###### 4.4.3. Summarize the results of simulation by plotting sankey diagram[¶](#4.4.3.-Summarize-the-results-of-simulation-by-plotting-sankey-diagram)

Sankey diagram is useful when you want to visualize cell transition between some groups.

plot\_mc\_resutls\_as\_sankey function can make a Sankey diagram with any arbitrary cluster information.

[89]:

# plot sankey diagram  
plt.figure(figsize=[5,6])  
cl = "louvain\_annot"  
oracle.plot\_mc\_resutls\_as\_sankey(cluster\_use=cl, start=0, end=100)

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_35_0.png>

The sankey diagram above looks messy because cluster order is random.

Change cluster order and make plot again

[90]:

cl = "louvain\_annot"  
order = ['MEP\_0', 'Mk\_0','Ery\_0', 'Ery\_1', 'Ery\_2', 'Ery\_3', 'Ery\_4',  
 'Ery\_5', 'Ery\_6', 'Ery\_7', 'Ery\_8', 'Ery\_9',  
 'GMP\_0', 'GMP\_1', 'GMP\_2', 'GMPl\_0', 'GMPl\_1',  
 'Mo\_0', 'Mo\_1', 'Mo\_2', 'Gran\_0', 'Gran\_1', 'Gran\_2', 'Gran\_3']  
  
plt.figure(figsize=[5,6])  
plt.subplots\_adjust(left=0.3, right=0.7)  
oracle.plot\_mc\_resutls\_as\_sankey(cluster\_use=cl, start=0, end=100, order=order)  
#plt.savefig(f"{save\_folder}/mcmc\_{cl}.png")

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_37_0.png>

Make another saneky diagram with different cluster unit.

[92]:

order = ['Megakaryocytes', 'MEP', 'Erythroids', 'GMP', 'late\_GMP', 'Monocytes', 'Granulocytes']  
cl = "cell\_type"  
  
plt.figure(figsize=[5,6])  
plt.subplots\_adjust(left=0.35, right=0.65)  
oracle.plot\_mc\_resutls\_as\_sankey(cluster\_use=cl, start=0, end=100, order=order, font\_size=14)  
#plt.savefig(f"{save\_folder}/mcmc\_{cl}{goi}.png", transparent=True)

<_images/notebooks_05_simulation_Gata1_KO_simulation_with_with_Paul_etal_2015_data_39_0.png>

In the example above, we may conclude several things as follows.

Gata1 KO induces cell state transition from Erythroids to MEP, from MEP to GMP. These results suggest that Gata1 may play important roles in the progression of Erythroid differentiation and cell state determination between MEP and GMP lineage.

The data also show that Gata1 KO may induce cell state transition from granulocytes to late GMP, suggesting Gata1’s involvement in Granulocytes differentiation.

These results agree with previous reports about Gata1. We could recapitulate Gata1’s cell-type-specific function regarding the cell fate decisions in hematopoiesis.

[ ]:

## API[¶](#api)

### Command Line API[¶](#command-line-api)

CellOracle have one command line API. This command can be used to convert scRNA-seq data. If you have a scRNA-seq data which was processed with seurat and saved as Rds file, you can use the following command to make anndata from seurat object. The anndata produced by this command can be used for an input of celloracle. Please go to step3 and step4 of the tutorial to see an example.

seuratToAnndata YOUR\_SEURAT\_OBJECT.Rds OUTPUT\_PATH

### Python API[¶](#python-api)

#### Custom class in celloracle[¶](#custom-class-in-celloracle)

We define some custom classes in celloracle.

*class* celloracle.Oracle[¶](#celloracle.Oracle)

Bases: celloracle.trajectory.modified\_VelocytoLoom\_class.modified\_VelocytoLoom

Oracle is the main class in CellOracle. Oracle object imports scRNA-seq data (anndata) and TF information to infer cluster-specific GRNs. It can predict future gene expression patterns and cell state transition after perturbations of TFs. Please see the paper of CellOracle for details.

The code of the Oracle class was made of three components below.

(1) Anndata: Gene expression matrix and metadata in single-cell RNA-seq are stored in anndata object. Processed values, such as normalized counts and simulated values, are stored as layers of anndata. Metadata (i.e., Cluster info) are saved in anndata.obs. Refer to scanpy/anndata documentation for detail.

(2) Net: Net is a custom class in celloracle. Net object process several data to infer GRN. See the documentation of Net class for detail.

(3) VelycytoLoom: Calculation of transition probability and visualization of directed trajectory graph will be performed in the same way as velocytoloom. VelocytoLoom is class for the Velocyto, which is a python library for RNA-velocity analysis. In the celloracle, we use almost the same functions as velocytoloom for the visualization, but celloracle use simulated gene expression values instead of RNA-velocity data.

Some CellOracle’s methods were inspired by velocyto analysis and some codes were made by modifying VelocytoLoom class.

adata[¶](#celloracle.Oracle.adata)

*anndata* – Imported anndata object

cluster\_column\_name[¶](#celloracle.Oracle.cluster_column_name)

*str* – The column name in adata.obs about cluster info

embedding\_name[¶](#celloracle.Oracle.embedding_name)

*str* – The key name in adata.obsm about dimensional reduction cordinates

addTFinfo\_dictionary(*TFdict*)[¶](#celloracle.Oracle.addTFinfo_dictionary)

Add new TF info to pre-existing TFdict. Values in the old TF dictionary will remain.

|  |  |
| --- | --- |
| Parameters: | **TFdict** (*dictionary*) – Python dictionary of TF info. |

copy()[¶](#celloracle.Oracle.copy)

Deepcopy itself.

fit\_GRN\_for\_simulation(*GRN\_unit='cluster'*, *alpha=1*, *use\_cluster\_specific\_TFdict=False*)[¶](#celloracle.Oracle.fit_GRN_for_simulation)

Do GRN inference. Please see the paper of CellOracle for details.

GRN can be constructed at an arbitrary cell group. If you want to infer cluster-specific GRN, please set [GRN\_unit=”cluster”]. GRN will be inferred for each cluster. You can select Cluster information when you import data (not when you run this method.).

If you set [GRN\_unit=”whole”], GRN will be made using all cells.

|  |  |
| --- | --- |
| Parameters: | * **GRN\_unit** (*str*) – select “cluster” or “whole” * **alpha** (*float* *or* *int*) – the strength of regularization. If you set a lower value, the sensitivity increase, and you can detect a weak network connection, but it might get more noize. With a higher value of alpha may reduce the chance of overfitting. |

get\_cluster\_specific\_TFdict\_from\_Links(*links\_object*)[¶](#celloracle.Oracle.get_cluster_specific_TFdict_from_Links)

Extract TF and its target gene information from Links object. This function can be used to reconstruct GRNs based on pre-existing GRNs saved in Links object.

|  |  |
| --- | --- |
| Parameters: | **links\_object** ([*Links*](index.html#celloracle.Links)) – Please see the explanation of Links class. |

get\_links(*cluster\_name\_for\_GRN\_unit=None*, *alpha=10*, *bagging\_number=20*, *verbose\_level=1*, *test\_mode=False*)[¶](#celloracle.Oracle.get_links)

Make GRN for each cluster and returns results as a Links object. Several preprocessing should be done before using this function.

|  |  |
| --- | --- |
| Parameters: | * **cluster\_name\_for\_GRN\_unit** (*str*) – Cluster name for GRN calculation. The cluster information should be stored in Oracle.adata.obs. * **alpha** (*float* *or* *int*) – the strength of regularization. If you set a lower value, the sensitivity increase, and you can detect a weak network connection, but it might get more noize. With a higher value of alpha may reduce the chance of overfitting. * **bagging\_number** (*int*) – The number for bagging calculation. * **verbose\_level** (*int*) – if [verbose\_level>1], most detailed progress information will be shown. if [verbose\_level > 0], one progress bar will be shown. if [verbose\_level == 0], no progress bar will be shown. * **test\_mode** (*bool*) – If test\_mode is True, GRN calculation will be done for only one cluster rather than all clusters. |

import\_TF\_data(*TF\_info\_matrix=None*, *TF\_info\_matrix\_path=None*, *TFdict=None*)[¶](#celloracle.Oracle.import_TF_data)

Load data about potential-regulatory TFs. You can import either TF\_info\_matrix or TFdict. See the tutorial of celloracle or motif\_analysis module for an example to make such files.

|  |  |
| --- | --- |
| Parameters: | * **TF\_info\_matrix** (*pandas.DataFrame*) – TF\_info\_matrix. * **TF\_info\_matrix\_path** (*str*) – File path for TF\_info\_matrix (pandas.DataFrame). * **TFdict** (*dictionary*) – Python dictionary of TF info. |

import\_anndata\_as\_normalized\_count(*adata*, *cluster\_column\_name=None*, *embedding\_name=None*)[¶](#celloracle.Oracle.import_anndata_as_normalized_count)

Load scRNA-seq data. scRNA-seq data should be prepared as an anndata. Preprocessing (cell and gene filtering, calculate DR and cluster, etc.) should be done before loading data. The method will import NORMALIZED and LOG TRANSFORMED but NOT SCALED and NOT CENTERED DATA. See the tutorial for the details for an example of how to process scRNA-seq data.

|  |  |
| --- | --- |
| Parameters: | * **adata** (*anndata*) – anndata object that store scRNA-seq data. * **cluster\_column\_name** (*str*) – the column name about cluster info in anndata.obs. Clustering data suppose to be in anndata.obs. * **embedding\_name** (*str*) – the key name about a dimensional reduction in anndata.obsm. Dimensional reduction (or 2D trajectory graph) should be in anndata.obsm. * **transform** (*str*) – The method for log-transformation. Chose one from “natural\_log” or “log2”. |

import\_anndata\_as\_raw\_count(*adata*, *cluster\_column\_name=None*, *embedding\_name=None*, *transform='natural\_log'*)[¶](#celloracle.Oracle.import_anndata_as_raw_count)

Load scRNA-seq data. scRNA-seq data should be prepared as an anndata. Preprocessing (cell and gene filtering, calculate DR and cluster, etc.) should be done before loading data. The method imports RAW GENE COUNTS because unscaled and uncentered gene expression data are required for the GRN inference and simulation. See tutorial notebook for the details about how to process scRNA-seq data.

|  |  |
| --- | --- |
| Parameters: | * **adata** (*anndata*) – anndata object that stores scRNA-seq data. * **cluster\_column\_name** (*str*) – the column name about cluster info in anndata.obs. Clustering data suppose to be in anndata.obs. * **embedding\_name** (*str*) – the key name about a dimensional reduction in anndata.obsm. Dimensional reduction (or 2D trajectory graph) should be in anndata.obsm. * **transform** (*str*) – The method for log-transformation. Chose one from “natural\_log” or “log2”. |

plot\_mc\_result\_as\_kde(*n\_time*, *args={}*)[¶](#celloracle.Oracle.plot_mc_result_as_kde)

Pick up one timepoint in the cell state-transition simulation and plot as a kde plot.

|  |  |
| --- | --- |
| Parameters: | * **n\_time** (*int*) – the number in Markov simulation * **args** (*dictionary*) – An argument for seaborn.kdeplot. See seaborn documentation for detail (<https://seaborn.pydata.org/generated/seaborn.kdeplot.html#seaborn.kdeplot>). |

plot\_mc\_result\_as\_trajectory(*cell\_name*, *time\_range*, *args={}*)[¶](#celloracle.Oracle.plot_mc_result_as_trajectory)

Pick up several timepoints in the cell state-transition simulation and plot as a line plot. This function can be used to visualize how cell-state changes after perturbation focusing on a specific cell.

|  |  |
| --- | --- |
| Parameters: | * **cell\_name** (*str*) – cell name. chose from adata.obs.index * **time\_range** (*list of int*) – the number in markov simulation * **args** (*dictionary*) – dictionary for the arguments for matplotlib.pyplit.plot. See matplotlib documentation for detail (<https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot>). |

plot\_mc\_resutls\_as\_sankey(*cluster\_use*, *start=0*, *end=-1*, *order=None*, *font\_size=10*)[¶](#celloracle.Oracle.plot_mc_resutls_as_sankey)

Plot the simulated cell state-transition as a Sankey-diagram after groping by the cluster.

|  |  |
| --- | --- |
| Parameters: | * **cluster\_use** (*str*) – cluster information name in anndata.obs. You can use any arbitrary cluster information in anndata.obs. * **start** (*int*) – The starting point of Sankey-diagram. Please select a step in the Markov simulation. * **end** (*int*) – The end point of Sankey-diagram. Please select a step in the Markov simulation. if you set [end=-1], the final step of Markov simulation will be used. * **order** (*list of str*) – The order of cluster name in sankey-diagram. * **font\_size** (*int*) – Font size for cluster name label in Sankey diagram. |

prepare\_markov\_simulation(*verbose=False*)[¶](#celloracle.Oracle.prepare_markov_simulation)

Pick up cells for Markov simulation.

|  |  |
| --- | --- |
| Parameters: | **verbose** (*bool*) – If True, it plots selected cells. |

run\_markov\_chain\_simulation(*n\_steps=500*, *n\_duplication=5*, *seed=123*)[¶](#celloracle.Oracle.run_markov_chain_simulation)

Do Markov simlation to predict cell transition after perturbation. The transition probability between cells has been calculated based on simulated gene expression values in the signal propagation process. The cell state transition will be simulated based on the probability. You can simulate the process for multiple times to get a robust outcome.

|  |  |
| --- | --- |
| Parameters: | * **n\_steps** (*int*) – steps for Markov simulation. This value is equivalent to the time after perturbation. * **n\_duplication** (*int*) – the number for multiple calculations. |

simulate\_shift(*perturb\_condition=None*, *GRN\_unit='whole'*, *n\_propagation=3*)[¶](#celloracle.Oracle.simulate_shift)

Simulate signal propagation with GRNs. Please see the paper of CellOracle for details. This function simulates a gene expression pattern in the near future. Simulated values will be stored in anndata.layers: [“simulated\_count”]

Three data below are used for the simulation. (1) GRN inference results (coef\_matrix). (2) perturb\_condition: You can set arbitrary perturbation condition. (3) gene expression matrix: simulation starts from imputed gene expression data.

|  |  |
| --- | --- |
| Parameters: | * **perturb\_condition** (*dictionary*) – condition for perturbation. if you want to simulate knockout for GeneX, please set [perturb\_condition={“GeneX”: 0.0}] Although you can set any non-negative values for the gene condition, avoid setting biologically unfeasible values for the perturb condition. It is strongly recommended to check actual gene expression values in your data before selecting perturb condition. * **n\_propagation** (*int*) – Calculation will be performed iteratively to simulate signal propagation in GRN. you can set the number of steps for this calculation. With a higher number, the results may recapitulate signal propagation for many genes. However, a higher number of propagation may cause more error/noise. |

summarize\_mc\_results\_by\_cluster(*cluster\_use*)[¶](#celloracle.Oracle.summarize_mc_results_by_cluster)

This function summarizes the simulated cell state-transition by groping the results into each cluster. It returns sumarized results as a pandas.DataFrame.

|  |  |
| --- | --- |
| Parameters: | **cluster\_use** (*str*) – cluster information name in anndata.obs. You can use any arbitrary cluster information in anndata.obs. |

to\_hdf5(*file\_path*)[¶](#celloracle.Oracle.to_hdf5)

Save object as hdf5.

|  |  |
| --- | --- |
| Parameters: | **file\_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.oracle’ |

updateTFinfo\_dictionary(*TFdict*)[¶](#celloracle.Oracle.updateTFinfo_dictionary)

Update a TF dictionary. If a key in the new TF dictionary already existed in the old TF dictionary, old values will be replaced with a new one.

|  |  |
| --- | --- |
| Parameters: | **TFdict** (*dictionary*) – Python dictionary of TF info. |

*class* celloracle.Links(*name*, *links\_dict={}*)[¶](#celloracle.Links)

Bases: object

This is a class for the processing and visualization of GRNs. Links object stores cluster-specific GRNs and metadata. Please use “get\_links” function in Oracle object to generate Links object.

links\_dict[¶](#celloracle.Links.links_dict)

*dictionary* – Dictionary that store unprocessed network data.

filtered\_links[¶](#celloracle.Links.filtered_links)

*dictionary* – Dictionary that store filtered network data.

merged\_score[¶](#celloracle.Links.merged_score)

*pandas.dataframe* – Network scores.

cluster[¶](#celloracle.Links.cluster)

*list of str* – List of cluster name.

name[¶](#celloracle.Links.name)

*str* – Name of clustering unit.

palette[¶](#celloracle.Links.palette)

*pandas.dataframe* – DataFrame that store color information.

add\_palette(*cluster\_name\_list*, *color\_list*)[¶](#celloracle.Links.add_palette)

Add color information for each cluster.

|  |  |
| --- | --- |
| Parameters: | * **cluster\_name\_list** (*list of str*) – cluster names. * **color\_list** (*list of stf*) – list of color for each cluster. |

filter\_links(*p=0.001*, *weight='coef\_abs'*, *thread\_number=10000*, *genelist\_source=None*, *genelist\_target=None*)[¶](#celloracle.Links.filter_links)

Filter network edges. In most case inferred GRN has non-significant random edges. We have to remove such random edges before analyzing network structure. You can do the filtering either of three ways below.

1. Do filtering based on the p-value of the network edge. Please enter p-value for thresholding.
2. Do filtering based on network edge number. If you set the number, network edges will be filtered based on the order of a network score. The top n-th network edges with network weight will remain, and the other edges will be removed. The network data have several types of network weight, so you have to select which network weight do you want to use.
3. Do filtering based on an arbitrary gene list. You can set a gene list for source nodes or target nodes.

|  |  |
| --- | --- |
| Parameters: | * **p** (*float*) – threshold for p-value of the network edge. * **weight** (*str*) – Please select network weight name for the filtering * **genelist\_source** (*list of str*) – gene list to remain in regulatory gene nodes. Default is None. * **genelist\_target** (*list of str*) – gene list to remain in target gene nodes. Default is None. |

get\_network\_entropy(*value='coef\_abs'*)[¶](#celloracle.Links.get_network_entropy)

Calculate network entropy scores. Please select a type of network weight for the entropy calculation.

|  |  |
| --- | --- |
| Parameters: | **value** (*str*) – Default is “coef\_abs”. |

get\_score(*test\_mode=False*)[¶](#celloracle.Links.get_score)

Get several network sores using R libraries. Make sure all dependant R libraries are installed in your environment before running this function. You can check the installation for the R libraries by running test\_installation() in network\_analysis module.

plot\_cartography\_scatter\_per\_cluster(*gois=None*, *clusters=None*, *scatter=False*, *kde=True*, *auto\_gene\_annot=False*, *percentile=98*, *args\_dot={}*, *args\_line={}*, *args\_annot={}*, *save=None*)[¶](#celloracle.Links.plot_cartography_scatter_per_cluster)

Make a plot of gene network cartography. Please read the original paper of gene network cartography for the principle of gene network cartography. <https://www.nature.com/articles/nature03288>

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **gois** (*list of srt*) – List of Gene name to highlight. * **clusters** (*list of str*) – List of cluster name to analyze. If None, all clusters in Links object will be analyzed. * **scatter** (*bool*) – Whether to make a scatter plot. * **auto\_gene\_annot** (*bool*) – Whether to pick up genes to make an annotation. * **percentile** (*float*) – Genes with a network score above the percentile will be shown with annotation. Default is 98. * **args\_dot** (*dictionary*) – Arguments for scatter plot. * **args\_line** (*dictionary*) – Arguments for lines in cartography plot. * **args\_annot** (*dictionary*) – Arguments for annotation in plots. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_cartography\_term(*goi*, *save=None*)[¶](#celloracle.Links.plot_cartography_term)

Plot the summary of gene network cartography like a heatmap. Please read the original paper of gene network cartography for the principle of gene network cartography. <https://www.nature.com/articles/nature03288>

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **gois** (*list of srt*) – List of Gene name to highlight. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_degree\_distributions(*plot\_model=False*, *save=None*)[¶](#celloracle.Links.plot_degree_distributions)

Plot the distribution of network degree (the number of edge per gene). The network degree will be visualized in both linear scale and log scale.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **plot\_model** (*bool*) – Whether to plot linear approximation line. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_network\_entropy\_distributions(*update\_network\_entropy=False*, *save=None*)[¶](#celloracle.Links.plot_network_entropy_distributions)

Plot the distribution of network entropy. See the CellOracle paper for the detail.

|  |  |
| --- | --- |
| Parameters: | * **links** (*Links object*) – See network\_analisis.Links class for detail. * **values** (*list of str*) – The list of score to visualize. If it is None, all network score (listed above) will be used. * **update\_network\_entropy** (*bool*) – Whether to recalculate network entropy. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_score\_comparison\_2D(*value*, *cluster1*, *cluster2*, *percentile=99*, *annot\_shifts=None*, *save=None*)[¶](#celloracle.Links.plot_score_comparison_2D)

Make a scatter plot that shows the relationship of a specific network score in two groups.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **value** (*srt*) – The network score type. * **cluster1** (*str*) – Cluster name. Network scores in the cluster1 will be visualized in the x-axis. * **cluster2** (*str*) – Cluster name. Network scores in the cluster2 will be visualized in the y-axis. * **percentile** (*float*) – Genes with a network score above the percentile will be shown with annotation. Default is 99. * **annot\_shifts** (*(float,* *float)*) – Annotation visualization setting. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_score\_discributions(*values=None*, *method='boxplot'*, *save=None*)[¶](#celloracle.Links.plot_score_discributions)

Plot the distribution of network scores. An individual data point is a network edge (gene) of GRN in each cluster.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See Links class for detail. * **values** (*list of str*) – The list of score to visualize. If it is None, all network score will be used. * **method** (*str*) – Plotting method. Select either of “boxplot” or “barplot”. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_score\_per\_cluster(*goi*, *save=None*)[¶](#celloracle.Links.plot_score_per_cluster)

Plot network score for a gene. This function visualizes the network score of a specific gene between clusters to get an insight into the dynamics of the gene.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **goi** (*srt*) – Gene name. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_scores\_as\_rank(*cluster*, *n\_gene=50*, *save=None*)[¶](#celloracle.Links.plot_scores_as_rank)

Pick up top n-th genes wich high-network scores and make plots.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **cluster** (*str*) – Cluster nome to analyze. * **n\_gene** (*int*) – Number of genes to plot. Default is 50. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

to\_hdf5(*file\_path*)[¶](#celloracle.Links.to_hdf5)

Save object as hdf5.

|  |  |
| --- | --- |
| Parameters: | **file\_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.links’ |

*class* celloracle.Net(*gene\_expression\_matrix*, *gem\_standerdized=None*, *TFinfo\_matrix=None*, *cellstate=None*, *TFinfo\_dic=None*, *annotation=None*, *verbose=True*)[¶](#celloracle.Net)

Bases: object

Net is a custom class for inferring sample-specific GRN from scRNA-seq data. This class is used inside Oracle class for GRN inference. This class requires two information below.

1. single-cell RNA-seq data The Net class needs processed scRNA-seq data. Gene and cell filtering, quality check, normalization, log-transformation (but not scaling and centering) have to be done before starting GRN calculation with this class. You can also use any arbitrary metadata (i.e., mRNA count, cell-cycle phase) for GRN input.
2. potential regulatory connection This method uses the list of potential regulatory TFs as input. This information can be calculated from Atac-seq data using the motif-analysis module. If sample-specific ATAC-seq data is not available, you can use general TF-binding info made from public ATAC-seq dataset of various tissue/cell type.

linkList[¶](#celloracle.Net.linkList)

*pandas.DataFrame* – the result of GRN inference.

all\_genes[¶](#celloracle.Net.all_genes)

*numpy.array* – an array of all genes that exist in input gene expression matrix

embedding\_name[¶](#celloracle.Net.embedding_name)

*str* – the key name name in adata.obsm about dimensional reduction coordinates

annotation[¶](#celloracle.Net.annotation)

*dictionary* – annotation. you can add an arbitrary annotation.

coefs\_dict[¶](#celloracle.Net.coefs_dict)

*dictionary* – coefs of linear regression.

stats\_dict[¶](#celloracle.Net.stats_dict)

*dictionary* – statistic values about coefs.

fitted\_genes[¶](#celloracle.Net.fitted_genes)

*list of str* – list of genes with which regression model was successfully calculated.

failed\_genes[¶](#celloracle.Net.failed_genes)

*list of str* – list of genes that were not able to get coefs

cellstate[¶](#celloracle.Net.cellstate)

*pandas.DataFrame* – metadata for GRN input

TFinfo[¶](#celloracle.Net.TFinfo)

*pandas.DataFrame* – information about potential regulatory TFs.

gem[¶](#celloracle.Net.gem)

*pandas.DataFrame* – merged matrix made with gene\_expression\_matrix and cellstate matrix.

gem\_standerdized[¶](#celloracle.Net.gem_standerdized)

*pandas.DataFrame* – almost the same as gem, but the gene\_expression\_matrix was standardized.

library\_last\_update\_date[¶](#celloracle.Net.library_last_update_date)

*str* – last update date of this code (this info is for code development. it will be deleted in the future)

object\_initiation\_date[¶](#celloracle.Net.object_initiation_date)

*str* – the date when this object is made.

addAnnotation(*annotation\_dictionary*)[¶](#celloracle.Net.addAnnotation)

Add a new annotation.

|  |  |
| --- | --- |
| Parameters: | **annotation\_dictionary** (*dictionary*) – e.g. {“sample\_name”: “NIH 3T3 cell”} |

addTFinfo\_dictionary(*TFdict*)[¶](#celloracle.Net.addTFinfo_dictionary)

Add a new TF info to pre-exiting TFdict.

|  |  |
| --- | --- |
| Parameters: | **TFdict** (*dictionary*) – python dictionary of TF info. |

addTFinfo\_matrix(*TFinfo\_matrix*)[¶](#celloracle.Net.addTFinfo_matrix)

Load TF info dataframe.

|  |  |
| --- | --- |
| Parameters: | **TFinfo** (*pandas.DataFrame*) – information about potential regulatory TFs. |

copy()[¶](#celloracle.Net.copy)

Deepcopy itself

fit\_All\_genes(*bagging\_number=200*, *scaling=True*, *model\_method='bagging\_ridge'*, *command\_line\_mode=False*, *log=None*, *alpha=1*, *verbose=True*)[¶](#celloracle.Net.fit_All_genes)

Make ML models for all genes. The calculation will be performed in parallel using scikit-learn bagging function. You can select a modeling method (bagging\_ridge or bayesian\_ridge). The calculation usually takes a long time.

|  |  |
| --- | --- |
| Parameters: | * **bagging\_number** (*int*) – The number of estimators for bagging. * **scaling** (*bool*) – Whether to scale regulatory gene expression values. * **model\_method** (*str*) – ML model name. “bagging\_ridge” or “bayesian\_ridge” * **command\_line\_mode** (*bool*) – Please select False if the calculation is performed on jupyter notebook. * **log** (*logging object*) – log object to output log * **alpha** (*int*) – Strength of regularization. * **verbose** (*bool*) – Whether to show a progress bar. |

fit\_All\_genes\_parallel(*bagging\_number=200*, *scaling=True*, *log=None*, *verbose=10*)[¶](#celloracle.Net.fit_All_genes_parallel)

!! This function has some bug now and currently unavailable.

Make ML models for all genes. The calculation will be performed in parallel using joblib parallel module.

|  |  |
| --- | --- |
| Parameters: | * **bagging\_number** (*int*) – The number of estimators for bagging. * **scaling** (*bool*) – Whether to scale regulatory gene expression values. * **log** (*logging object*) – log object to output log * **verbose** (*int*) – verbose for joblib parallel |

fit\_genes(*target\_genes*, *bagging\_number=200*, *scaling=True*, *model\_method='bagging\_ridge'*, *save\_coefs=False*, *command\_line\_mode=False*, *log=None*, *alpha=1*, *verbose=True*)[¶](#celloracle.Net.fit_genes)

Make ML models for genes of interest. The calculation will be performed in parallel using scikit-learn bagging function. You can select a modeling method (bagging\_ridge or bayesian\_ridge).

|  |  |
| --- | --- |
| Parameters: | * **target\_genes** (*list of str*) – gene list * **bagging\_number** (*int*) – The number of estimators for bagging. * **scaling** (*bool*) – Whether to scale regulatory gene expression values. * **model\_method** (*str*) – ML model name. “bagging\_ridge” or “bayesian\_ridge” * **save\_coefs** (*bool*) – Whether to store details of coef values in bagging model. * **command\_line\_mode** (*bool*) – Please select False if the calculation is performed on jupyter notebook. * **log** (*logging object*) – log object to output log * **alpha** (*int*) – Strength of regularization. * **verbose** (*bool*) – Whether to show a progress bar. |

plotCoefs(*target\_gene*, *sort=True*, *threshold\_p=None*)[¶](#celloracle.Net.plotCoefs)

Plot the distribution of Coef values (network edge weights).

|  |  |
| --- | --- |
| Parameters: | * **target\_gene** (*str*) – gene name * **sort** (*bool*) – Whether to sort genes by its strength * **bagging\_number** (*int*) – The number of estimators for bagging. * **threshold\_p** (*float*) – the threshold for p-values. TFs will be filtered based on the p-value. if None, no filtering is applied. |

to\_hdf5(*file\_path*)[¶](#celloracle.Net.to_hdf5)

Save object as hdf5.

|  |  |
| --- | --- |
| Parameters: | **file\_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.net’ |

updateLinkList(*verbose=True*)[¶](#celloracle.Net.updateLinkList)

Update linkList. LinkList is a data frame that store information about inferred GRN.

|  |  |
| --- | --- |
| Parameters: | **verbose** (*bool*) – Whether to show a progress bar |

updateTFinfo\_dictionary(*TFdict*)[¶](#celloracle.Net.updateTFinfo_dictionary)

Update TF info matrix

|  |  |
| --- | --- |
| Parameters: | **TFdict** (*dictionary*) – A python dictionary in which a key is Target gene, value are potential regulatory genes for the target gene. |

celloracle.load\_hdf5(*file\_path*, *object\_class\_name=None*)[¶](#celloracle.load_hdf5)

Load an object of celloracle’s custom class that was saved as hdf5.

|  |  |
| --- | --- |
| Parameters: | * **file\_path** (*str*) – file\_path. * **object\_class\_name** (*str*) – Types of object. If it is None, object class will be identified from the extension of file\_name. Default is None. |

#### Modules for ATAC-seq analysis[¶](#modules-for-atac-seq-analysis)

##### celloracle.motif\_analysis module[¶](#module-celloracle.motif_analysis)

The [motif\_analysis](#module-celloracle.motif_analysis) module implements transcription factor motif scan.

Genomic activity information (peak of ATAC-seq or Chip-seq) is extracted first. Then the peak DNA sequence will be subjected to TF motif scan. Finally we will get list of TFs that potentially binds to a specific gene.

celloracle.motif\_analysis.is\_genome\_installed(*ref\_genome*)[¶](#celloracle.motif_analysis.is_genome_installed)

Celloracle motif\_analysis module uses gimmemotifs and genomepy internally. Reference genome files should be installed in the PC to use gimmemotifs and genomepy. This function checks the installation status of the reference genome.

|  |  |
| --- | --- |
| Parameters: | **ref\_genome** (*str*) – names of reference genome. i.e., “mm10”, “hg19” |

celloracle.motif\_analysis.peak2fasta(*peak\_ids*, *ref\_genome*)[¶](#celloracle.motif_analysis.peak2fasta)

Convert peak\_id into fasta object.

|  |  |
| --- | --- |
| Parameters: | * **peak\_id** (*str* *or* *list of str*) – Peak\_id. e.g. “chr5\_0930303\_9499409” or it can be a list of peak\_id. e.g. [“chr5\_0930303\_9499409”, “chr11\_123445555\_123445577”] * **ref\_genome** (*str*) – Reference genome name. e.g. “mm9”, “mm10”, “hg19” etc |
| Returns: | DNA sequence in fasta format |
| Return type: | gimmemotifs fasta object |

celloracle.motif\_analysis.read\_bed(*bed\_path*)[¶](#celloracle.motif_analysis.read_bed)

Load bed file and return as dataframe.

|  |  |
| --- | --- |
| Parameters: | **bed\_path** (*str*) – File path. |
| Returns: | bed file in dataframe. |
| Return type: | pandas.dataframe |

celloracle.motif\_analysis.load\_TFinfo\_from\_parquets(*folder\_path*)[¶](#celloracle.motif_analysis.load_TFinfo_from_parquets)

Load TFinfo object which was saved with the function; “save\_as\_parquet”.

|  |  |
| --- | --- |
| Parameters: | **folder\_path** (*str*) – folder path |
| Returns: | Loaded TFinfo object. |
| Return type: | [TFinfo](index.html#celloracle.motif_analysis.TFinfo) |

celloracle.motif\_analysis.make\_TFinfo\_from\_scanned\_file(*path\_to\_raw\_bed*, *path\_to\_scanned\_result\_bed*, *ref\_genome*)[¶](#celloracle.motif_analysis.make_TFinfo_from_scanned_file)

This function is currently an available.

*class* celloracle.motif\_analysis.TFinfo(*peak\_data\_frame*, *ref\_genome*)[¶](#celloracle.motif_analysis.TFinfo)

Bases: object

This is a custom class for motif analysis in celloracle. TFinfo object performs motif scan using the TF motif database in gimmemotifs and several functions of genomepy. Analysis results can be exported as a python dictionary or dataframe. These files; python dictionary of dataframe of TF binding information, are needed in GRN inference.

peak\_df[¶](#celloracle.motif_analysis.TFinfo.peak_df)

*pandas.dataframe* – dataframe about DNA peak and target gene data.

all\_target\_gene[¶](#celloracle.motif_analysis.TFinfo.all_target_gene)

*array of str* – target genes.

ref\_genome[¶](#celloracle.motif_analysis.TFinfo.ref_genome)

*str* – reference genome name that was used in DNA peak generation.

scanned\_df[¶](#celloracle.motif_analysis.TFinfo.scanned_df)

*dictionary* – Results of motif scan. Key is a peak name. Value is a dataframe of motif scan.

dic\_targetgene2TFs[¶](#celloracle.motif_analysis.TFinfo.dic_targetgene2TFs)

*dictionary* – Final product of Motif scan. Key is a target gene. Value is a list of regulatory candidate genes.

dic\_peak2Targetgene[¶](#celloracle.motif_analysis.TFinfo.dic_peak2Targetgene)

*dictionary* – Dictionary. Key is a peak name. Value is a list of the target gene.

dic\_TF2targetgenes[¶](#celloracle.motif_analysis.TFinfo.dic_TF2targetgenes)

*dictionary* – Final product of Motif scan. Key is a TF. Value is a list of potential target genes of the TF.

copy()[¶](#celloracle.motif_analysis.TFinfo.copy)

Deepcoty itself.

filter\_motifs\_by\_score(*threshold*, *method='cumlative\_score'*)[¶](#celloracle.motif_analysis.TFinfo.filter_motifs_by_score)

Remove motifs with low binding scores.

|  |  |
| --- | --- |
| Parameters: | **method** (*str*) – thresholding method. Select either of [“indivisual\_score”, “cumlative\_score”] |

filter\_peaks(*peaks\_to\_be\_remained*)[¶](#celloracle.motif_analysis.TFinfo.filter_peaks)

Filter peaks.

|  |  |
| --- | --- |
| Parameters: | **peaks\_to\_be\_remained** (*array of str*) – list of peaks. Peaks that are NOT in the list will be removed. |

make\_TFinfo\_dataframe\_and\_dictionary(*verbose=True*)[¶](#celloracle.motif_analysis.TFinfo.make_TFinfo_dataframe_and_dictionary)

This is the final step of motif\_analysis. Convert scanned results into a data frame and dictionaries.

|  |  |
| --- | --- |
| Parameters: | **verbose** (*bool*) – Whether to show a progress bar. |

reset\_dictionary\_and\_df()[¶](#celloracle.motif_analysis.TFinfo.reset_dictionary_and_df)

Reset TF dictionary and TF dataframe. The following attributes will be erased; TF\_onehot, dic\_targetgene2TFs, dic\_peak2Targetgene, dic\_TF2targetgenes.

reset\_filtering()[¶](#celloracle.motif_analysis.TFinfo.reset_filtering)

Reset filtering information. You can use this function to stat over the filtering step with new conditions. The following attributes will be erased; TF\_onehot, dic\_targetgene2TFs, dic\_peak2Targetgene, dic\_TF2targetgenes.

save\_as\_parquet(*folder\_path=None*)[¶](#celloracle.motif_analysis.TFinfo.save_as_parquet)

Save itself. Some attributes are saved as parquet file.

|  |  |
| --- | --- |
| Parameters: | **folder\_path** (*str*) – folder path |

scan(*background\_length=200*, *fpr=0.02*, *n\_cpus=-1*, *verbose=True*)[¶](#celloracle.motif_analysis.TFinfo.scan)

Scan DNA sequences searching for TF binding motifs.

|  |  |
| --- | --- |
| Parameters: | * **background\_length** (*int*) – background length. This is used for the calculation of the binding score. * **fpr** (*float*) – False positive rate for motif identification. * **n\_cpus** (*int*) – number of CPUs for parallel calculation. * **verbose** (*bool*) – Whether to show a progress bar. |

to\_dataframe(*verbose=True*)[¶](#celloracle.motif_analysis.TFinfo.to_dataframe)

Return results as a dataframe. Rows are peak\_id, and columns are TFs.

|  |  |
| --- | --- |
| Parameters: | **verbose** (*bool*) – Whether to show a progress bar. |
| Returns: | TFinfo matrix. |
| Return type: | pandas.dataframe |

to\_dictionary(*dictionary\_type='targetgene2TFs'*, *verbose=True*)[¶](#celloracle.motif_analysis.TFinfo.to_dictionary)

Return TF information as a python dictionary.

|  |  |
| --- | --- |
| Parameters: | **dictionary\_type** (*str*) – Type of dictionary. Select from [“targetgene2TFs”, “TF2targetgenes”]. If you chose “targetgene2TFs”, it returns a dictionary in which a key is a target gene, and a value is a list of regulatory candidate genes (TFs) of the target. If you chose “TF2targetgenes”, it returns a dictionary in which a key is a TF and a value is a list of potential target genes of the TF. |
| Returns: | dictionary. |
| Return type: | dictionary |

to\_hdf5(*file\_path*)[¶](#celloracle.motif_analysis.TFinfo.to_hdf5)

Save object as hdf5.

|  |  |
| --- | --- |
| Parameters: | **file\_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.tfinfo’ |

celloracle.motif\_analysis.get\_tss\_info(*peak\_str\_list*, *ref\_genome*, *verbose=True*)[¶](#celloracle.motif_analysis.get_tss_info)

Get annotation about Transcription Starting Site (TSS).

|  |  |
| --- | --- |
| Parameters: | * **peak\_str\_list** (*list of str*) – list of peak\_id. e.g., [“chr5\_0930303\_9499409”, “chr11\_123445555\_123445577”] * **ref\_genome** (*str*) – reference genome name. * **verbose** (*bool*) – verbosity. |

celloracle.motif\_analysis.integrate\_tss\_peak\_with\_cicero(*tss\_peak*, *cicero\_connections*)[¶](#celloracle.motif_analysis.integrate_tss_peak_with_cicero)

Process output of cicero data and returns DNA peak information for motif analysis in celloracle. Please see the tutorial of celloracle documentation.

|  |  |
| --- | --- |
| Parameters: | * **tss\_peak** (*pandas.dataframe*) – dataframe about TSS information. Please use the function, “get\_tss\_info” to get this dataframe. * **cicero\_connections** (*dataframe*) – dataframe that stores the results of cicero analysis. |
| Returns: | DNA peak about promoter/enhancer and its annotation about target gene. |
| Return type: | pandas.dataframe |

#### Modules for Network analysis[¶](#modules-for-network-analysis)

##### celloracle.network\_analysis module[¶](#module-celloracle.network_analysis)

The [network\_analysis](#module-celloracle.network_analysis) module implements Network analysis.

celloracle.network\_analysis.get\_links(*oracle\_object*, *cluster\_name\_for\_GRN\_unit=None*, *alpha=10*, *bagging\_number=20*, *verbose\_level=1*, *test\_mode=False*)[¶](#celloracle.network_analysis.get_links)

Make GRN for each cluster and returns results as a Links object. Several preprocessing should be done before using this function.

|  |  |
| --- | --- |
| Parameters: | * **oracle\_object** ([*Oracle*](index.html#celloracle.Oracle)) – See Oracle module for detail. * **cluster\_name\_for\_GRN\_unit** (*str*) – Cluster name for GRN calculation. The cluster information should be stored in Oracle.adata.obs. * **alpha** (*float* *or* *int*) – the strength of regularization. If you set a lower value, the sensitivity increase, and you can detect a weak network connection, but it might get more noize. With a higher value of alpha may reduce the chance of overfitting. * **bagging\_number** (*int*) – The number for bagging calculation. * **verbose\_level** (*int*) – if [verbose\_level>1], most detailed progress information will be shown. if [verbose\_level > 0], one progress bar will be shown. if [verbose\_level == 0], no progress bar will be shown. * **test\_mode** (*bool*) – If test\_mode is True, GRN calculation will be done for only one cluster rather than all clusters. |

celloracle.network\_analysis.test\_R\_libraries\_installation()[¶](#celloracle.network_analysis.test_R_libraries_installation)

CellOracle.network\_analysis use some R libraries for network analysis. This is a test function to check instalation of necessary R libraries.

celloracle.network\_analysis.load\_links(*file\_path*)[¶](#celloracle.network_analysis.load_links)

Load links object saved as a hdf5 file.

|  |  |
| --- | --- |
| Parameters: | **file\_path** (*str*) – file path. |
| Returns: | loaded links object. |
| Return type: | [Links](index.html#celloracle.Links) |

*class* celloracle.network\_analysis.Links(*name*, *links\_dict={}*)[¶](#celloracle.network_analysis.Links)

Bases: object

This is a class for the processing and visualization of GRNs. Links object stores cluster-specific GRNs and metadata. Please use “get\_links” function in Oracle object to generate Links object.

links\_dict[¶](#celloracle.network_analysis.Links.links_dict)

*dictionary* – Dictionary that store unprocessed network data.

filtered\_links[¶](#celloracle.network_analysis.Links.filtered_links)

*dictionary* – Dictionary that store filtered network data.

merged\_score[¶](#celloracle.network_analysis.Links.merged_score)

*pandas.dataframe* – Network scores.

cluster[¶](#celloracle.network_analysis.Links.cluster)

*list of str* – List of cluster name.

name[¶](#celloracle.network_analysis.Links.name)

*str* – Name of clustering unit.

palette[¶](#celloracle.network_analysis.Links.palette)

*pandas.dataframe* – DataFrame that store color information.

add\_palette(*cluster\_name\_list*, *color\_list*)[¶](#celloracle.network_analysis.Links.add_palette)

Add color information for each cluster.

|  |  |
| --- | --- |
| Parameters: | * **cluster\_name\_list** (*list of str*) – cluster names. * **color\_list** (*list of stf*) – list of color for each cluster. |

filter\_links(*p=0.001*, *weight='coef\_abs'*, *thread\_number=10000*, *genelist\_source=None*, *genelist\_target=None*)[¶](#celloracle.network_analysis.Links.filter_links)

Filter network edges. In most case inferred GRN has non-significant random edges. We have to remove such random edges before analyzing network structure. You can do the filtering either of three ways below.

1. Do filtering based on the p-value of the network edge. Please enter p-value for thresholding.
2. Do filtering based on network edge number. If you set the number, network edges will be filtered based on the order of a network score. The top n-th network edges with network weight will remain, and the other edges will be removed. The network data have several types of network weight, so you have to select which network weight do you want to use.
3. Do filtering based on an arbitrary gene list. You can set a gene list for source nodes or target nodes.

|  |  |
| --- | --- |
| Parameters: | * **p** (*float*) – threshold for p-value of the network edge. * **weight** (*str*) – Please select network weight name for the filtering * **genelist\_source** (*list of str*) – gene list to remain in regulatory gene nodes. Default is None. * **genelist\_target** (*list of str*) – gene list to remain in target gene nodes. Default is None. |

get\_network\_entropy(*value='coef\_abs'*)[¶](#celloracle.network_analysis.Links.get_network_entropy)

Calculate network entropy scores. Please select a type of network weight for the entropy calculation.

|  |  |
| --- | --- |
| Parameters: | **value** (*str*) – Default is “coef\_abs”. |

get\_score(*test\_mode=False*)[¶](#celloracle.network_analysis.Links.get_score)

Get several network sores using R libraries. Make sure all dependant R libraries are installed in your environment before running this function. You can check the installation for the R libraries by running test\_installation() in network\_analysis module.

plot\_cartography\_scatter\_per\_cluster(*gois=None*, *clusters=None*, *scatter=False*, *kde=True*, *auto\_gene\_annot=False*, *percentile=98*, *args\_dot={}*, *args\_line={}*, *args\_annot={}*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_cartography_scatter_per_cluster)

Make a plot of gene network cartography. Please read the original paper of gene network cartography for the principle of gene network cartography. <https://www.nature.com/articles/nature03288>

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **gois** (*list of srt*) – List of Gene name to highlight. * **clusters** (*list of str*) – List of cluster name to analyze. If None, all clusters in Links object will be analyzed. * **scatter** (*bool*) – Whether to make a scatter plot. * **auto\_gene\_annot** (*bool*) – Whether to pick up genes to make an annotation. * **percentile** (*float*) – Genes with a network score above the percentile will be shown with annotation. Default is 98. * **args\_dot** (*dictionary*) – Arguments for scatter plot. * **args\_line** (*dictionary*) – Arguments for lines in cartography plot. * **args\_annot** (*dictionary*) – Arguments for annotation in plots. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_cartography\_term(*goi*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_cartography_term)

Plot the summary of gene network cartography like a heatmap. Please read the original paper of gene network cartography for the principle of gene network cartography. <https://www.nature.com/articles/nature03288>

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **gois** (*list of srt*) – List of Gene name to highlight. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_degree\_distributions(*plot\_model=False*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_degree_distributions)

Plot the distribution of network degree (the number of edge per gene). The network degree will be visualized in both linear scale and log scale.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **plot\_model** (*bool*) – Whether to plot linear approximation line. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_network\_entropy\_distributions(*update\_network\_entropy=False*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_network_entropy_distributions)

Plot the distribution of network entropy. See the CellOracle paper for the detail.

|  |  |
| --- | --- |
| Parameters: | * **links** (*Links object*) – See network\_analisis.Links class for detail. * **values** (*list of str*) – The list of score to visualize. If it is None, all network score (listed above) will be used. * **update\_network\_entropy** (*bool*) – Whether to recalculate network entropy. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_score\_comparison\_2D(*value*, *cluster1*, *cluster2*, *percentile=99*, *annot\_shifts=None*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_score_comparison_2D)

Make a scatter plot that shows the relationship of a specific network score in two groups.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **value** (*srt*) – The network score type. * **cluster1** (*str*) – Cluster name. Network scores in the cluster1 will be visualized in the x-axis. * **cluster2** (*str*) – Cluster name. Network scores in the cluster2 will be visualized in the y-axis. * **percentile** (*float*) – Genes with a network score above the percentile will be shown with annotation. Default is 99. * **annot\_shifts** (*(float,* *float)*) – Annotation visualization setting. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_score\_discributions(*values=None*, *method='boxplot'*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_score_discributions)

Plot the distribution of network scores. An individual data point is a network edge (gene) of GRN in each cluster.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See Links class for detail. * **values** (*list of str*) – The list of score to visualize. If it is None, all network score will be used. * **method** (*str*) – Plotting method. Select either of “boxplot” or “barplot”. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_score\_per\_cluster(*goi*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_score_per_cluster)

Plot network score for a gene. This function visualizes the network score of a specific gene between clusters to get an insight into the dynamics of the gene.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **goi** (*srt*) – Gene name. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

plot\_scores\_as\_rank(*cluster*, *n\_gene=50*, *save=None*)[¶](#celloracle.network_analysis.Links.plot_scores_as_rank)

Pick up top n-th genes wich high-network scores and make plots.

|  |  |
| --- | --- |
| Parameters: | * **links** ([*Links*](index.html#celloracle.Links)) – See network\_analisis.Links class for detail. * **cluster** (*str*) – Cluster nome to analyze. * **n\_gene** (*int*) – Number of genes to plot. Default is 50. * **save** (*str*) – Folder path to save plots. If the folder does not exist in the path, the function creates the folder. Plots will not be saved if [save=None]. Default is None. |

to\_hdf5(*file\_path*)[¶](#celloracle.network_analysis.Links.to_hdf5)

Save object as hdf5.

|  |  |
| --- | --- |
| Parameters: | **file\_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.links’ |

celloracle.network\_analysis.transfer\_scores\_from\_links\_to\_adata(*adata*, *links*, *method='median'*)[¶](#celloracle.network_analysis.transfer_scores_from_links_to_adata)

Transfer the summary of network scores (median or mean) per group from Links object into adata.

|  |  |
| --- | --- |
| Parameters: | * **adata** (*anndata*) – anndata * **links** ([*Links*](index.html#celloracle.Links)) – likns object * **method** (*str*) – The method to summarize data. |

celloracle.network\_analysis.linkList\_to\_networkgraph(*filteredlinkList*)[¶](#celloracle.network_analysis.linkList_to_networkgraph)

Convert linkList into Graph object in NetworkX.

|  |  |
| --- | --- |
| Parameters: | **filteredlinkList** (*pandas.DataFrame*) – GRN saved as linkList. |
| Returns: | Network X graph objenct. |
| Return type: | Graph object |

celloracle.network\_analysis.draw\_network(*linkList*, *return\_graph=False*)[¶](#celloracle.network_analysis.draw_network)

Plot network graph.

|  |  |
| --- | --- |
| Parameters: | * **linkList** (*pandas.DataFrame*) – GRN saved as linkList. * **return\_graph** (*bool*) – Whether to return graph object. |
| Returns: | Network X graph objenct. |
| Return type: | Graph object |

#### Other modules[¶](#other-modules)

##### celloracle.go\_analysis module[¶](#celloracle-go-analysis-module)

The [go\_analysis](#module-celloracle.go_analysis) module implements Gene Ontology analysis. This module use goatools internally.

celloracle.go\_analysis.geneSymbol2ID(*symbols*, *species='mouse'*)[¶](#celloracle.go_analysis.geneSymbol2ID)

Convert gene symbol into Entrez gene id.

|  |  |
| --- | --- |
| Parameters: | * **symbols** (*array of str*) – gene symbol * **species** (*str*) – Select species. Either “mouse” or “human” |
| Returns: | Entrez gene id |
| Return type: | list of str |

celloracle.go\_analysis.geneID2Symbol(*IDs*, *species='mouse'*)[¶](#celloracle.go_analysis.geneID2Symbol)

Convert Entrez gene id into gene symbol.

|  |  |
| --- | --- |
| Parameters: | * **IDs** (*array of str*) – Entrez gene id. * **species** (*str*) – Select species. Either “mouse” or “human”. |
| Returns: | Gene symbol |
| Return type: | list of str |

celloracle.go\_analysis.get\_GO(*gene\_query*, *species='mouse'*)[¶](#celloracle.go_analysis.get_GO)

Get Gene Ontologies (GOs).

|  |  |
| --- | --- |
| Parameters: | * **gene\_query** (*array of str*) – gene list. * **species** (*str*) – Select species. Either “mouse” or “human” |
| Returns: | GO analysis results as dataframe. |
| Return type: | pandas.dataframe |

##### celloracle.utility module[¶](#celloracle-utility-module)

The [utility](#module-celloracle.utility) module has several functions that support celloracle.

*class* celloracle.utility.makelog(*file\_name=None*, *directory=None*)[¶](#celloracle.utility.makelog)

Bases: object

This is a class for making log.

info(*comment*)[¶](#celloracle.utility.makelog.info)

Add comment into the log file.

|  |  |
| --- | --- |
| Parameters: | **comment** (*str*) – comment. |

celloracle.utility.save\_as\_pickled\_object(*obj*, *filepath*)[¶](#celloracle.utility.save_as_pickled_object)

Save any object using pickle.

|  |  |
| --- | --- |
| Parameters: | * **obj** (*any python object*) – python object. * **filepath** (*str*) – file path. |

celloracle.utility.load\_pickled\_object(*filepath*)[¶](#celloracle.utility.load_pickled_object)

Load pickled object.

|  |  |
| --- | --- |
| Parameters: | **filepath** (*str*) – file path. |
| Returns: | loaded object. |
| Return type: | python object |

celloracle.utility.intersect(*list1*, *list2*)[¶](#celloracle.utility.intersect)

Intersect two list and get components that exists in both list.

|  |  |
| --- | --- |
| Parameters: | * **list1** (*list*) – input list. * **list2** (*list*) – input list. |
| Returns: | intersected list. |
| Return type: | list |

celloracle.utility.exec\_process(*commands*, *message=True*, *wait\_finished=True*, *return\_process=True*)[¶](#celloracle.utility.exec_process)

Excute a command. This is a wrapper of “subprocess.Popen”

|  |  |
| --- | --- |
| Parameters: | * **commands** (*str*) – command. * **message** (*bool*) – Whether to return a message or not. * **wait\_finished** (*bool*) – Whether to wait for the process finished. If False, the function finish immediately. * **return\_process** (*bool*) – Whether to return “process”. |

celloracle.utility.standard(*df*)[¶](#celloracle.utility.standard)

Standerdize value.

|  |  |
| --- | --- |
| Parameters: | **df** (*padas.dataframe*) – dataframe. |
| Returns: | data after standerdization. |
| Return type: | pandas.dataframe |

celloracle.utility.load\_hdf5(*file\_path*, *object\_class\_name=None*)[¶](#celloracle.utility.load_hdf5)

Load an object of celloracle’s custom class that was saved as hdf5.

|  |  |
| --- | --- |
| Parameters: | * **file\_path** (*str*) – file\_path. * **object\_class\_name** (*str*) – Types of object. If it is None, object class will be identified from the extension of file\_name. Default is None. |

celloracle.utility.inverse\_dictionary(*dictionary*, *verbose=True*, *return\_value\_as\_numpy=False*)[¶](#celloracle.utility.inverse_dictionary)

Make inversed dictionary. See examples below for detail.

|  |  |
| --- | --- |
| Parameters: | * **dictionary** (*dict*) – python dictionary * **verbose** (*bool*) – Whether to show progress bar. * **return\_value\_as\_numpy** (*bool*) – Whether to convert values into numpy array. |
| Returns: | Python dictionary. |
| Return type: | dict |

Examples

>>> dic = {"a": [1, 2, 3], "b": [2, 3, 4]}  
>>> inverse\_dictionary(dic)  
{1: ['a'], 2: ['a', 'b'], 3: ['a', 'b'], 4: ['b']}

>>> dic = {"a": [1, 2, 3], "b": [2, 3, 4]}  
>>> inverse\_dictionary(dic, return\_value\_as\_numpy=True)  
{1: array(['a'], dtype='<U1'),  
 2: array(['a', 'b'], dtype='<U1'),  
 3: array(['a', 'b'], dtype='<U1'),  
 4: array(['b'], dtype='<U1')}

##### celloracle.data module[¶](#celloracle-data-module)

The [data](#module-celloracle.data) module implements data download and loading.

celloracle.data.load\_TFinfo\_df\_mm9\_mouse\_atac\_atlas()[¶](#celloracle.data.load_TFinfo_df_mm9_mouse_atac_atlas)

Load Transcription factor binding information made from mouse scATAC-seq atlas dataset. mm9 genome was used for the reference genome.

Args:

|  |  |
| --- | --- |
| Returns: | TF binding info. |
| Return type: | pandas.dataframe |

##### celloracle.data\_conversion module[¶](#celloracle-data-conversion-module)

The [data\_conversion](#module-celloracle.data_conversion) module implements data conversion between different platform.

celloracle.data\_conversion.seurat\_object\_to\_anndata(*file\_path\_seurat\_object*, *delete\_tmp\_file=True*)[¶](#celloracle.data_conversion.seurat_object_to_anndata)

Convert seurat object into anndata.

|  |  |
| --- | --- |
| Parameters: | * **file\_path\_seurat\_object** (*str*) – File path of seurat object. Seurat object should be saved as Rds format. * **delete\_tmp\_file** (*bool*) – Whether to delete temporary file. |
| Returns: | anndata object. |
| Return type: | anndata |

## Changelog[¶](#changelog)

## License[¶](#license)

The software is provided under xxxxxxx License.

License xxxxxxxx  
  
Copyright (c) 2019 Kenji Kamimoto, Christy Hoffmann, Samantha Morris  
  
  
This is a placeholder. License description will be shown here.

## Authors and citations[¶](#authors-and-citations)

### Cite celloracle[¶](#cite-celloracle)

If you use velocyto please cite our bioarxiv preprint TITLE.

### celloracle software development[¶](#celloracle-software-development)

celloracle is developed and maintained by the effort of celloracle team in Samantha Morris Lab. Please post troubles or questions on [the Github repository](https://cole-trapnell-lab.github.io/cicero-release/)

# Indices and tables[¶](#indices-and-tables)

* [Index](genindex.html)
* [Module Index](py-modindex.html)
* [Search Page](search.html)

© Copyright 2019, Samantha Morris Lab

Built with [Sphinx](http://sphinx-doc.org/) using a [theme](https://github.com/rtfd/sphinx_rtd_theme) provided by [Read the Docs](https://readthedocs.org).