
celloracle

Release 0.5.1

Samantha Morris Lab

Aug 06, 2020

CONTENTS

1	Contents	3
1.1	Installation	3
1.2	Tutorial	7
1.3	API	114
1.4	Changelog	136
1.5	License	137
1.6	Authors and citations	140
2	Indices and tables	141
	Python Module Index	143
	Index	145

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

Source code is available at [celloracle GitHub repository](#)

For more information, please read our bioarxiv preprint: [CellOracle: Dissecting cell identity via network inference and in silico gene perturbation](#)

Note:

Documentation is also available as a pdf file.

[pdf documentation](#)

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

CONTENTS

1.1 Installation

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

1.1.1 Docker image

- Not available now. Coming soon.

1.1.2 System Requirements

- Operating system: macOS or linux are highly recommended. `celloracle` was developed and tested in Linux and macOS.
- We found that the `celloracle` calculation may be EXTREMELY SLOW under an environment of Windows Subsystem for Linux (WSL). We do not recommend using WSL.
- While you can install `celloracle` in Windows OS, please do so at your own risk and responsibility. We DO NOT provide any support for the use in the Windows OS.
- Memory: 8 G byte or more. Memory usage also depends on your scRNA-seq data. Especially in silico perturbation requires large amount of memory.
- CPU: Core i5 or better processor. GRN inference supports multicore calculation. Higher number of CPU cores enables faster calculation.

1.1.3 Python Requirements

- `celloracle` was developed with python 3.6. We do not support python 2.7x or python <=3.5.
- Please install all dependent libraries before installing `celloracle` according to the instructions below.
- `celloracle` 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

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

```
conda create -n celloracle_env python=3.6
conda activate celloracle_env
```

1. Add conda channels

Installation of some libraries requires 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

Please install velocyto with the following commands or the author's instruction .

```
conda install numpy scipy cython numba matplotlib scikit-learn h5py click pysam llvml
↳louvain
```

Then

```
pip install velocyto
```

It was reported that some compile errors might occur during the installation of velocyto on MacOS. Various errors were reported and you need to find the best solution depending on your error. You may find the solution with these links below.

- Solution 1: Install Xcode. Please try this first.
- Solution 2: Install `macOS_SDK_headers`. This solution is needed in addition to Solution-1 if your OS is MacOS Mojave.
- Solution 3. This is the solution reported by a CellOracle user. Thank you very much!
- Other solutions on [Velocyto github issue page](#)

3. Install scanpy

Please install scanpy with the following commands or the author's instruction .

```
conda install scanpy
```

4. Install other python libraries

Please install other python libraries below with the following commands.

```
conda install goatools pyarrow tqdm joblib jupyter
```

5. install celloracle from github

```
pip install git+https://github.com/morris-lab/CellOracle.git
```

1.1.4 R requirements

celloracle use R libraries for the network analysis and scATAC-seq analysis. Please install [R](#) (≥ 3.5) and R libraries below according to the author's instruction.

Seurat

Please install Seurat with the following r-script or [the author's instruction](#). 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

Please install Cicero and Monocle3 with the following r-script or [the author's instruction](#). If you do not have scATAC-seq data and plan to use celloracle's base GRN, you do not need to install Cicero.

In R console,

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(c("Gviz", "GenomicRanges", "rtracklayer"))

install.packages("devtools")
devtools::install_github("cole-trapnell-lab/cicero-release", ref = "monocle3")
```

igraph

Please install igraph with the following r-script or [the author's instruction](#).

In R console,

```
install.packages("igraph")
```

linkcomm

Please install `linkcomm` with the following r-script or the author's instruction .

In R console,

```
install.packages("linkcomm")
```

rnetcarto

Please install `rnetcarto` with the following r-script or the author's instruction .

In R console,

```
install.packages("rnetcarto")
```

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.

R path: /usr/lib/R/bin/R

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

The first line above is your R path. If you want to use another R program that was installed at the different place, you can set new R path with the following command.

```
co.network_analysis.set_R_path("ENTER YOUR R PATH HERE")
```

If you changed R path settings, please check installation again to make sure everything works.

```
co.network_analysis.test_R_libraries_installation()
```

1.2 Tutorial

The analysis proceeds through multiple steps. Please run the notebooks sequentially. If you do not have ATAC-seq data and want to use the default TF binding information, you can skip the first and second step and start from the third step.

Please refer to the `celloracle` paper for scientific premise and the detail of the algorithm of celloracle.

The jupyter notebook files and data used in this tutorial are available [here](#).

1.2.1 ATAC-seq data preprocessing

In this step, we process scATAC-seq data (or bulk ATAC-seq data) to obtain the accessible promoter/enhancer DNA sequence. We can get the active proximal promoter/enhancer genome sequences by picking up the ATAC-seq peaks that exist around the transcription starting site (TSS). Distal cis-regulatory elements can be picked up using `Cicero`. Cicero analyzes scATAC-seq data to calculate a co-accessible score between peaks. We can identify cis-regulatory elements using Cicero's co-access score and TSS information.

If you have bulk ATAC-seq data instead of scATAC-data, we'll get only the proximal promoter/enhancer genome sequences.

A. Extract TF binding information from scATAC-seq data

If you have scATAC-seq data, you can get information on the distal cis-regulatory elements. This step uses Cicero and does not use celloracle. You need to get co-accessibility table in this analysis. Although we provide an example notebook here, you can analyze your data with Cicero in a different way if you are familiar with Cicero. If you have a question about Cicero, please read the documentation of `Cicero` for the detailed usage.

scATAC-seq analysis with Cicero and Monocle3

The jupyter notebook files and data used in this tutorial are available [here](#).

R notebook

This is an example R script for Cicero analysis. In this R notebook, we'll use Cicero and Monocle3.

Please make sure that you installed these packages in advance.

You can download notebook file and additional data files from celloracle github page. https://github.com/morris-lab/CellOracle/tree/master/docs/notebooks/01_ATAC-seq_data_processing/option1_scATAC-seq_data_analysis_with_cicero

Another tutorial notebook that uses Monocle2 is also available in the celloracle github page above.

0. Import library

```
[2]: library(cicero)
library(monocle3)
```

1. Prepare data

In this tutorial we'll use acATAC-seq data from the 10x genomics database. You do not need to download these data if you analyze your own scATAC-seq 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]: # You can substitute the data path below with the data path of your scATAC data.
data_folder <- "data/filtered_peak_bc_matrix"

# Create a folder to save results
output_folder <- "cicero_output"
dir.create(output_folder)
```

2. Load data and make Cell Data Set (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(new_cell_data_set(indata,
cell_metadata = cellinfo,
gene_metadata = peakinfo))

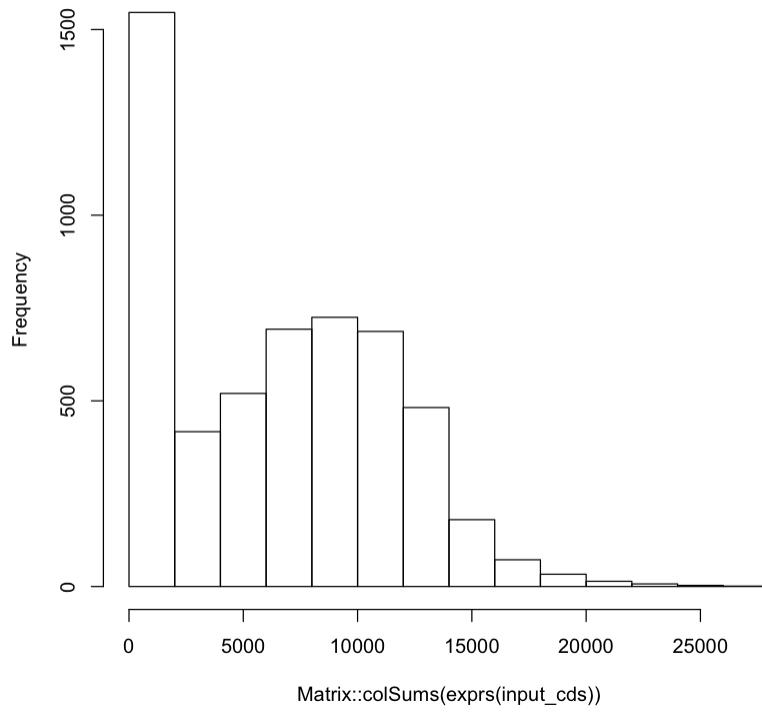
input_cds <- monocle3::detect_genes(input_cds)

#Ensure there are no peaks included with zero reads
input_cds <- input_cds[Matrix:::rowSums(exprs(input_cds)) != 0,]
```

3. Quality check and Filtering

```
[8]: # Visualize peak_count_per_cell
hist(Matrix::colSums(exprs(input_cds)))
```

Histogram of Matrix::colSums(exprs(input_cds))



```
[9]: # filter cells by peak_count
# PLEASE SET APPROPRIATE THRESHOLD VALUES according to your data
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

```
[ ]: # Data preprocessing
set.seed(2017)

input_cds <- detect_genes(input_cds)
input_cds <- estimate_size_factors(input_cds)
input_cds <- preprocess_cds(input_cds, method = "LSI")

# Dimensional reduction with umap
input_cds <- reduce_dimension(input_cds, reduction_method = 'UMAP',
                                preprocess_method = "LSI")
```

(continues on next page)

(continued from previous page)

```
umap_coords <- reducedDims(input_cds)$UMAP

cicero_cds <- make_cicero_cds(input_cds, reduced_coordinates = umap_coords)

# Save cds object if you want
saveRDS(cicero_cds, paste0(output_folder, "/cicero_cds.Rds"))
```

5. Load reference genome information

To run cicero, we need to get a genome coordinates files, which contains the lengths of each chromosomes. You can read mm10 genome information with the following command. The text file, mm10_chromosome_length.txt is included in the celloracle notebook folder.

https://github.com/morris-lab/CellOracle/tree/master/docs/notebooks/01_ATAC-seq_data_processing/option1_scATAC-seq_data_analysis_with_cicero

If your scATAC-seq data use different reference genome, you need to get a genome coordinates files for your reference genome. Please see the Cicero documentation for more information.

https://cole-trapnell-lab.github.io/cicero-release/docs_m3/#installing-cicero

```
[ ]: # !!Please make sure that the reference genome information below match the reference_
  ↪genome of your scATAC-seq data.

# If your scATAC-seq uses mm10 reference genome, you can read chromosome length file_
  ↪with the following command.
chromosome_length <- read.table("./mm10_chromosome_length.txt")

# For mm9 genome, you can use the following command.
#data("mouse.mm9.genome")
#chromosome_length <- mouse.mm9.genome

# For hg19 genome, you can use the following command.
#data("human.hg19.genome")
#chromosome_length <- mhuman.hg19.genome
```

6. Run Cicero

```
[11]: # run the main function
conns <- run_cicero(cicero_cds, 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"
```

(continues on next page)

(continued from previous page)

```
[1] "Running models"
[1] "Assembling connections"
[1] "Done"
```

	Peak1 <fct>	Peak2 <fct>	coaccess <dbl>
A data.frame: 6 × 3	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 the 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"))
```

TSS annotation

The jupyter notebook files and data used in this tutorial are available [here](#).

Python notebook

In this notebook, we process the results of cicero analysis to get active promoter/enhancer DNA peaks. First, we pick up peaks around the 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 so that the final product includes TSS peaks and peaks that have a strong connection with the TSS peaks. We use this information as an active promoter/enhancer elements.

0. Import libraries

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import os, sys, shutil, importlib, glob
from tqdm.notebook import tqdm

from celloracle import motif_analysis as ma
```

```
[2]: %config InlineBackend.figure_format = 'retina'

plt.rcParams['figure.figsize'] = [6, 4.5]
plt.rcParams["savefig.dpi"] = 300
```

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

IMPORTANT: Please make sure that you are setting correct reference genome.

If your scATAC-seq data was generated with mm10 reference genome, you can set ref_genome="mm10". If you used hg19 human reference genome, please set ref_genome=="hg19"

Currently we support refgenomes below. {"Human": ["hg38", "hg19"], "Mouse": ["mm10", "mm9"], "S.cerevisiae": ["sacCer2", "sacCer3"]}

If your reference genome is not in the list, please send a request through github issue page.

```
[5]: tss_annotated = ma.get_tss_info(peak_str_list=peaks, ref_genome= ) ####! Set reference
  ↪genome here

# 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        S1c25a26      +
16984  chr19   45659222  45660823        Fbxw4        -
16985  chr12   100898848 100899597        Gpr68        -
16986  chr4    129491262 129492047        Fam229a      -
```

3. Integrate TSS info and cicero connections

The output file after the integration process has three columns; “peak_id”, “gene_short_name”, and “coaccess”. “peak_id” is either the TSS peak or the peaks that have a connection with the TSS peak. “gene_short_name” is the gene name that associated with the TSS site. “coaccess” is the co-access score between a peak and TSS peak. Note, the TSS peak is indicated by a score of 1.

```
[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

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

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

Bulk DNA-seq data can be used to get the accessible promoter/enhancer sequences.

The jupyter notebook files and data used in this tutorial are available [here](#).

Python notebook

0. Import libraries

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import os, sys, shutil, importlib, glob
from tqdm import tqdm_notebook as tqdm

%config InlineBackend.figure_format = 'retina'

plt.rcParams['figure.figsize'] = [6, 4.5]
plt.rcParams["savefig.dpi"] = 300
```

```
[2]: # Import celloracle function
from celloracle import motif_analysis as ma
```

1. Load bed file

Import ATAC-seq bed file. This script can also be used with 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

IMPORTANT: Please make sure that you are setting the correct ref genome!

```
[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

```
[10]: tss_annotated.to_parquet("peak_file.parquet")
```

-> go to next notebook

1.2.2 Transcription factor binding motif scan

We identified accessible Promoter/enhancer DNA regions using ATAC-seq data. Next, we will obtain a list of TFs for each target gene by scanning the regulatory genomic sequences for TF-binding motifs. In the later GRN inference process, this list will be used to define potential regulatory connections.

The jupyter notebook files and data used in this tutorial are available [here](#).

Scan DNA sequences searching for TF binding motifs

Python notebook

0. Import libraries

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns

import os, sys, shutil, importlib, glob
from tqdm.notebook import tqdm
```



```
[2]: from celloracle import motif_analysis as ma
from celloracle.utility import save_as_pickled_object
```



```
[3]: %config InlineBackend.figure_format = 'retina'
%matplotlib inline

plt.rcParams['figure.figsize'] = (15, 7)
plt.rcParams["savefig.dpi"] = 600
```

1. Reference genome data preparation

1.1. Check reference genome installation

Celloracle uses genomepy to get DNA sequence data. Before starting celloracle analysis, we need to make sure that the reference genome is correctly installed in your computational environment. If not, please install reference genome.

```
[4]: # PLEASE make sure that you are setting correct ref genome.
ref_genome = "mm10"

genome_installation = ma.is_genome_installed(ref_genome=ref_genome)
print(ref_genome, "installation:", genome_installation)

mm10 installation: True
```

1.2. Install reference genome (if refgenome is not installed)

```
[5]: if not genome_installation:
    import genomepy
    genomepy.install_genome(ref_genome, "UCSC")
else:
    print(ref_genome, "is installed.")

mm10 is installed.
```

2. Load data

2.1. Load processed peak data

```
[6]: # Load annotated peak data.
peaks = pd.read_parquet("../01_ATAC-seq_data_processing/option1_scATAC-seq_data_"
                       "analysis_with_cicero/peak_file.parquet")
peaks.head()

[6]:
          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
```

2.1. Check data

```
[7]: # Define function for quality check
def decompose_chrstr(peak_str):
    """
    Args:
        peak_str (str): peak_str. e.g. 'chr1_3094484_3095479'

    Returns:
        tuple: chromosome name, start position, end position
    """
    *chr_, start, end = peak_str.split("_")
    chr_ = "_".join(chr_)
    return chr_, start, end

from genomepy import Genome

def check_peak_fomat(peaks_df, ref_genome):
    """
    Check peak fomat.
    (1) Check chromosome name.
    (2) Check peak size (length) and remove sort DNAs (<5bp)

    """
    df = peaks_df.copy()

    n_peaks_before = df.shape[0]

    # Decompose peaks and make df
    decomposed = [decompose_chrstr(peak_str) for peak_str in df["peak_id"]]
    df_decomposed = pd.DataFrame(np.array(decomposed))
    df_decomposed.columns = ["chr", "start", "end"]
    df_decomposed["start"] = df_decomposed["start"].astype(np.int)
    df_decomposed["end"] = df_decomposed["end"].astype(np.int)

    # Load genome data
    genome_data = Genome(ref_genome)
```

(continues on next page)

(continued from previous page)

```

all_chr_list = list(genome_data.keys())

# DNA length check
lengths = np.abs(df_decomposed["end"] - df_decomposed["start"])

# Filter peaks with invalid chromosome name
n_threshold = 5
df = df[(lengths >= n_threshold) & df_decomposed.chr.isin(all_chr_list)]

# DNA length check
lengths = np.abs(df_decomposed["end"] - df_decomposed["start"])

# Data counting
n_invalid_length = len(lengths[lengths < n_threshold])
n_peaks_invalid_chr = n_peaks_before - df_decomposed.chr.isin(all_chr_list).sum()
n_peaks_after = df.shape[0]

#
print("Peaks before filtering: ", n_peaks_before)
print("Peaks with invalid chr_name: ", n_peaks_invalid_chr)
print("Peaks with invalid length: ", n_invalid_length)
print("Peaks after filtering: ", n_peaks_after)

return df

```

[8]:

```

peaks = check_peak_fomat(peaks, ref_genome)

Peaks before filtering: 15680
Peaks with invalid chr_name: 0
Peaks with invalid length: 0
Peaks after filtering: 15680

```

2.2. [Optional step] Load motifs

You can select TF binding motif data for Celloracle motif analysis. If you have no preference and just want to use a default motif, you can skip this step. If you want to use a non-default motif dataset, please prepare motif data as a list of motif class in gimmermotifs. We have several option for loading motif database as below.

2.2.1. [Optional step] Load motif data from gimmermotifs dataset

Many motif databases are included with GimmeMotifs. <https://gimmermotifs.readthedocs.io/en/master/overview.html>
You can load them as follows.

[9]:

```

# First, we need to pick up motifs for your dataset. We can get a list.

# Get folder path that stores motif data.
import os, glob
from gimmermotifs.motif import MotifConfig
config = MotifConfig()
motif_dir = config.get_motif_dir()

```

(continues on next page)

(continued from previous page)

```
# Get list for motif data name
motifs_data_name = [i for i in os.listdir(motif_dir) if i.endswith(".pfm")]
motifs_data_name.sort()
motifs_data_name
```

[9]:

```
['CIS-BP.pfm',
 'ENCODE.pfm',
 'HOCOMOCov10_HUMAN.pfm',
 'HOCOMOCov10_MOUSE.pfm',
 'HOCOMOCov11_HUMAN.pfm',
 'HOCOMOCov11_MOUSE.pfm',
 'HOMER.pfm',
 'IMAGE.pfm',
 'JASPAR2018.pfm',
 'JASPAR2018_fungi.pfm',
 'JASPAR2018_insects.pfm',
 'JASPAR2018_nematodes.pfm',
 'JASPAR2018_plants.pfm',
 'JASPAR2018_urochordates.pfm',
 'JASPAR2018_vertebrates.pfm',
 'JASPAR2020.pfm',
 'JASPAR2020_fungi.pfm',
 'JASPAR2020_insects.pfm',
 'JASPAR2020_nematodes.pfm',
 'JASPAR2020_plants.pfm',
 'JASPAR2020_urochordates.pfm',
 'JASPAR2020_vertebrates.pfm',
 'RSAT_insects.pfm',
 'RSAT_plants.pfm',
 'RSAT_vertebrates.pfm',
 'SwissRegulon.pfm',
 'factorbook.pfm',
 'gimme.vertebrate.v5.0.pfm']
```

[10]:

```
# Once you picked up motifs, you can load the motif files with "read_motifs"
from gimmermotifs.motif import read_motifs

path = os.path.join(motif_dir, "JASPAR2018_plants.pfm") # Please enter motifs name here
motifs = read_motifs(path)

# Check first 10 motifs
motifs[:10]
```

[10]:

```
[MA0020.1_Dof2_AAAGCn,
 MA0021.1_Dof3_AAAGyn,
 MA0034.1_Gam1_nnyAACCGmC,
 MA0044.1_HMG-1_sTTGTnyTy,
 MA0045.1_HMG-I/Y_nwAnAAAnrnmrAmAy,
 MA0053.1_MNB1A_AAAGC,
 MA0054.1_myb.Ph3_TAACnGTTw,
 MA0064.1_PBF_AAAGY,
 MA0082.1_squamosa_mC AwAwATrGwAAn,
 MA0096.1_bZIP910_mTGACGT]
```

2.2.2 [Optional step] Load motif data from celloracle motif dataset

Celloracle also provides many motif dataset that was generated from CisBP. <http://cisbp.ccbr.utoronto.ca/>

These motifs were organized by each species. Please select motifs for your species.

If you have a request for motifs for a new species, you can ask us to add new motifs through github issue page.

```
[11]: # Check available motifs
ma.MOTIFS_LIST
```

```
[11]: ['CisBP_ver2_Arabidopsis_thaliana.pfm',
'CisBP_ver2_Caenorhabditis_elegans.pfm',
'CisBP_ver2_Danio rerio.pfm',
'CisBP_ver2_Drosophila_ananassae.pfm',
'CisBP_ver2_Drosophila_erecta.pfm',
'CisBP_ver2_Drosophila_grimshawi.pfm',
'CisBP_ver2_Drosophila_melanogaster.pfm',
'CisBP_ver2_Drosophila_mix.pfm',
'CisBP_ver2_Drosophila_mojavensis.pfm',
'CisBP_ver2_Drosophila_persimilis.pfm',
'CisBP_ver2_Drosophila_pseudoobscura.pfm',
'CisBP_ver2_Drosophila_sechellia.pfm',
'CisBP_ver2_Drosophila_simulans.pfm',
'CisBP_ver2_Drosophila_virilis.pfm',
'CisBP_ver2_Drosophila_willistoni.pfm',
'CisBP_ver2_Drosophila_yakuba.pfm',
'CisBP_ver2_Homo_sapiens.pfm',
'CisBP_ver2_Mus_musculus.pfm',
'CisBP_ver2_Rattus_norvegicus.pfm',
'CisBP_ver2_Saccharomyces_cerevisiae.pfm',
'CisBP_ver2_Xenopus_laevis.pfm',
'CisBP_ver2_Xenopus_tropicalis.pfm',
'CisBP_ver2_Xenopus_tropicalis_and_Xenopus_laevis.pfm']
```

```
[12]: # Load motifs from celloracle dataset.
motifs = ma.load_motifs("CisBP_ver2_Mus_musculus.pfm")

# Check first 10 motifs
motifs[:10]
```

```
[12]: [M00008_2.00_nnnAAww,
M00044_2.00_nrTAAACAn,
M00056_2.00_TAATAAAT,
M00060_2.00_nnnTTCnnn,
M00111_2.00_nGCCynnGGs,
M00112_2.00_CCTsrGGCnA,
M00113_2.00_nsCCnnAGGs,
M00114_2.00_nnGCCynnGG,
M00115_2.00_nnATnAAAn,
M00116_2.00_nnAATATTAnn]
```

3. Instantiate TFinfo object and search for TF binding motifs

The motif analysis module has a custom class; TFinfo. The TFinfo object converts a peak data into a DNA sequences and scans the DNA sequences searching for TF binding motifs. Then, the results of motif scan will be filtered and converted into either a python dictionary or a depending on your preference. This TF information is necessary for GRN inference.

3.1. Instantiate TFinfo object

```
[16]: # Instantiate TFinfo object
tfi = ma.TFinfo(peak_data_frame=peaks, # peak info calculated from ATAC-seq data
                 ref_genome=ref_genome)
```

3.2. Motif scan

!!You can set TF binding motif information as an argument:

```
tfi.scan(motifs=motifs)
```

If you don't set motifs or set None, default motifs will be loaded automatically.

- For mouse and human, “gimme.vertebrate.v5.0.” will be used as a default motifs.
- For another species, a species specific TF binding motif data extracted from CisBP ver2.0 will be used.

```
[ ]: %%time
# Scan motifs. !!CAUTION!! This step may take several hours if you have many peaks!
tfi.scan(fpr=0.02,
          motifs=None, # If you enter None, default motifs will be loaded.
          verbose=True)

# Save tfinfo object
tfi.to_hdf5(file_path="test1.celloracle.tfinfo")
```

```
[16]: # Check motif scan results
tfi.scanned_df.head()
```

	seqname	motif_id	factors_direct	\
0	chr10_100015291_100017830	GM.5.0.Homeodomain.0001	TGIF1	
1	chr10_100015291_100017830	GM.5.0.Mixed.0001		
2	chr10_100015291_100017830	GM.5.0.Mixed.0001		
3	chr10_100015291_100017830	GM.5.0.Mixed.0001		
4	chr10_100015291_100017830	GM.5.0.Nuclear_receptor.0002	NR2C2	

	factors_indirect	score	pos	strand
0	ENSG00000234254, TGIF1	10.311002	1003	1
1	SRF, EGR1	7.925873	481	1
2	SRF, EGR1	7.321375	911	-1
3	SRF, EGR1	7.276585	811	-1
4	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.

4. Filtering motifs

```
[15]: # 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: 12952283 -> 2288874
1. converting scanned results into one-hot encoded dataframe.

HBox(children=(FloatProgress(value=0.0, max=14142.0), HTML(value='')))

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

HBox(children=(FloatProgress(value=0.0, max=15006.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=1090.0), HTML(value='')))
```

5. Get Final results

5.1. Get results as a dictionary

```
[23]: td = tfi.to_dictionary(dictionary_type="targetgene2TFs")
```

5.2. Get results as a dataframe

```
[17]: df = tfi.to_dataframe()
df.head()

[17]:          peak_id gene_short_name  9430076c15rik  Ac002126.6 \
0  chr10_100015291_100017830           Kitl        0.0        0.0
1  chr10_100486534_100488209         Tmtc3        0.0        0.0
2  chr10_100588641_100589556  4930430F08Rik        0.0        0.0
3  chr10_100741247_100742505         Gm35722        0.0        0.0
4  chr10_101681379_101682124        Mgat4c        0.0        0.0

          Ac012531.1  Ac226150.2   Afp    Ahr   Ahrr   Aire   ...   Znf784   Znf8   Znf816 \
0            0.0      0.0  0.0  1.0  1.0  0.0   ...        0.0  0.0  0.0
1            0.0      0.0  0.0  0.0  0.0  0.0   ...        1.0  0.0  0.0
2            1.0      0.0  0.0  1.0  1.0  0.0   ...        0.0  0.0  0.0
3            0.0      0.0  0.0  0.0  0.0  0.0   ...        0.0  0.0  0.0
4            0.0      0.0  0.0  0.0  0.0  0.0   ...        0.0  0.0  0.0
```

(continues on next page)

(continued from previous page)

	Znf85	Zscan10	Zscan16	Zscan22	Zscan26	Zscan31	Zscan4
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0

[5 rows x 1092 columns]

6. Save TFinfo as dictionary or dataframe

We'll use this information when making the GRNs. Save the results.

```
[19]: 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"))
```

[Optional step 1] How to use different motif data

Celloracle motif analysis pipeline provides several default motifs. If you don't enter motif data, celloracle automatically load default motifs for your species. In that case, you don't need to make TF binding motifs. But also, you can pick up TF binding motifs by yourself. Here, we introduce how to find and load motifs for celloracle motif analysis. Some codes for custom motif were based on the suggestion in [this post](#). from KyleFerchen. Thank you Kyle!

0. Overview: How to use a different motifs for celloracle motif scan.

In this notebook, we introduce how to prepare motif dataset for celloracle motif analysis. Celloracle uses list of motif object in `gimmemotifs` package. See `gimmemotifs` documentation for more details. (<https://gimmemotifs.readthedocs.io/en/master/api.html#>)

1 Import motifs from `gimmemotifs` dataset.

`Gimmemotifs` provides many motif dataset that was generated from public motif database including CisDB, ENCODE, HOMER, and JASPAR. <https://gimmemotifs.readthedocs.io/en/master/overview.html>

1.1 gimme.vertebrate.v5.0.

By default GimmeMotifs uses a non-redundant, clustered database of known vertebrate motifs. These motifs come from CIS-BP (<http://cisbp.ccbr.utoronto.ca/>) and other sources. This motif dataset can be easily loaded with the following command.

If your dataset is Mouse or Human, this one will be a good default choice.

```
[1]: # Compare with default motifs in gimmemotifs
from gimmemotifs.motif import default_motifs
motifs = default_motifs()

# Check first 10 motifs
motifs[:10]
```



```
[1]: [GM.5.0.Sox.0001_AACAAT,
GM.5.0.Homeodomain.0001_AGCTGTCAnA,
GM.5.0.Mixed.0001_snnGGssGGs,
GM.5.0.Nuclear_receptor.0001_TAwstrGGTCAsTrGGTCA,
GM.5.0.Mixed.0002_GCTAATTA,
GM.5.0.Nuclear_receptor.0002_wnyrCTTCCGGGkC,
GM.5.0.bHLH.0001_ACGTG,
GM.5.0.Myb_SANT.0001_rrCCGTTAACnGyy,
GM.5.0.C2H2_ZF.0001_GCGkGGGCGG,
GM.5.0.GATA.0001_TTATCTsnnnnnnnCA]
```

1.2 Motifs that are provided with gimmemotifs package

Many other motif databases come included with GimmeMotifs. You can load them as follows.

```
[2]: # Get folder path that stores motif data.
import os, glob
from gimmemotifs.motif import MotifConfig
config = MotifConfig()
motif_dir = config.get_motif_dir()

# Get motif data names
motifs_data_name = [i for i in os.listdir(motif_dir) if i.endswith(".pfm")]
motifs_data_name.sort()
motifs_data_name
```



```
[2]: ['CIS-BP.pfm',
'ENCODE.pfm',
'HOCOMOCOv10_HUMAN.pfm',
'HOCOMOCOv10_MOUSE.pfm',
'HOCOMOCOv11_HUMAN.pfm',
'HOCOMOCOv11_MOUSE.pfm',
'HOMER.pfm',
'IMAGE.pfm',
'JASPAR2018.pfm',
'JASPAR2018_fungi.pfm',
'JASPAR2018_insects.pfm',
'JASPAR2018_nematodes.pfm',
'JASPAR2018_plants.pfm',
'JASPAR2018_urochordates.pfm',
'JASPAR2018_vertebrates.pfm',
'JASPAR2020.pfm',
```

(continues on next page)

(continued from previous page)

```
'JASPAR2020_fungi.pfm',
'JASPAR2020_insects.pfm',
'JASPAR2020_nematodes.pfm',
'JASPAR2020_plants.pfm',
'JASPAR2020_urochordates.pfm',
'JASPAR2020_vertebrates.pfm',
'RSAT_insects.pfm',
'RSAT_plants.pfm',
'RSAT_vertebrates.pfm',
'SwissRegulon.pfm',
'factorbook.pfm',
'gimme.vertebrate.v5.0.pfm']
```

```
[3]: # You can load motif files with "read_motifs"
from gimmermotifs.motif import read_motifs

path = os.path.join(motif_dir, "JASPAR2018_plants.pfm")
motifs = read_motifs(path)

# Check first 10 motifs
motifs[:10]
```

```
[3]: [MA0020.1_Dof2_AAAGCn,
MA0021.1_Dof3_AAAGyn,
MA0034.1_Gam1_nnyAACCGmC,
MA0044.1_HMG-1_sTTGTnyTy,
MA0045.1_HMG-I/Y_nwAnAAAnrnmrAmAy,
MA0053.1_MNB1A_AAAGC,
MA0054.1_myb.Ph3_TAACnGTTw,
MA0064.1_PBF_AAAGY,
MA0082.1_squamosa_mC AwAwATrGwAAn,
MA0096.1_bZIP910_mTGACGT]
```

2. Import motifs from Celloracle dataset.

Celloracle provides many motif dataset that was generated from CisDB. These motifs were divided by species. Please select motifs for your species.

```
[4]: from celloracle import motif_analysis as ma

# Check available motifs
ma.MOTIFS_LIST
```

```
[4]: ['CisBP_ver2_Arabidopsis_thaliana.pfm',
'CisBP_ver2_Caenorhabditis_elegans.pfm',
'CisBP_ver2_Danio rerio.pfm',
'CisBP_ver2_Drosophila_ananassae.pfm',
'CisBP_ver2_Drosophila_erecta.pfm',
'CisBP_ver2_Drosophila_grimshawi.pfm',
'CisBP_ver2_Drosophila_melanogaster.pfm',
'CisBP_ver2_Drosophila_mix.pfm',
'CisBP_ver2_Drosophila_mojavensis.pfm',
'CisBP_ver2_Drosophila_persimilis.pfm',
'CisBP_ver2_Drosophila_pseudoobscura.pfm',
'CisBP_ver2_Drosophila_sechellia.pfm']
```

(continues on next page)

(continued from previous page)

```
'CisBP_ver2_Drosophila_simulans.pfm',
'CisBP_ver2_Drosophila_virilis.pfm',
'CisBP_ver2_Drosophila_willistoni.pfm',
'CisBP_ver2_Drosophila_yakuba.pfm',
'CisBP_ver2_Homo_sapiens.pfm',
'CisBP_ver2_Mus_musculus.pfm',
'CisBP_ver2_Rattus_norvegicus.pfm',
'CisBP_ver2_Saccharomyces_cerevisiae.pfm',
'CisBP_ver2_Xenopus_laevis.pfm',
'CisBP_ver2_Xenopus_tropicalis.pfm',
'CisBP_ver2_Xenopus_tropicalis_and_Xenopus_laevis.pfm']
```

```
[5]: # Load motifs from celloracle dataset.
motifs = ma.load_motifs("CisBP_ver2_Homo_sapiens.pfm")

# Check first 10 motifs
motifs[:10]

[5]: [M00056_2.00_TAATAAAT,
M00070_2.00_nrAACAAATAnn,
M00111_2.00_nGCCyynnGGs,
M00112_2.00_CCTsrGGCnA,
M00113_2.00_nsCCnAGGs,
M00114_2.00_nnGCCynnGG,
M00115_2.00_nnATnAAAn,
M00116_2.00_nnAATATTAnn,
M00130_2.00_nnnGCCCGnn,
M00142_2.00_GTrCTCmy]
```

3. Import motifs from custom motif dataset.

If you want to use another motif data source, you need to make a list of motif class in `gimmemotifs`. The easiest way to make such object is to use “`read_motifs`” function in `gimmemotifs`.

This function can load motif data text file. You need to prepare two files, `XXX.motif2factors.txt` and `XXX.pfm`.

3.1 XXX.motif2factors.txt

The text file, `XXX.motif2factors.txt` includes TF factor annotation for each motifs. The file should be like a tsv file like below.

The first column should be motif name, the motif name should match with motif name in `pfm` file. The second column is gene symbol, the third column is datasource. This column is not important. The forth column is additional information for this factor. The factor is labeled with “Y” If factor information was confirmed by some evidence. Otherwise, the factor is labeled with “N”.

```
[52]: path_motif2factors = path.replace(".pfm", ".motif2factors.txt")

with open(path_motif2factors, "r") as f:
    for i, j in enumerate(f):
        print(j)
        if i>5:
            break
```

Motif	Factor	Evidence	Curated
MA0020.1_Dof2	Dof2	SELEX	Y
MA0021.1_Dof3	Dof3	SELEX	Y
MA0034.1_Gam1	Gam1	SELEX	Y
MA0044.1_HMG-1	HMG-1	SELEX	Y
MA0045.1_HMG-I/Y		HMG-I/Y SELEX	Y
MA0053.1_MNB1A	MNB1A	SELEX	Y

3.2 XXX.pfm

The second file, XXX.pfm. should includes motif pwm information. The file shoud be like below.

The motif name in this pfm file should exactly match with the motif name in motif2factor.txt file.

```
[55]: with open(path, "r") as f:
    for i, j in enumerate(f):
        print(j)
        if i>10:
            break

# JASPAR2018_plants motif database

# Retrieved from: http://jaspar.genereg.net/download/CORE/JASPAR2018_CORE_plants_non-
# redundant_pfms_jaspar.txt

# Date: 2018-10-17

>MA0020.1_Dof2

0.9999 0.0000 0.0000 0.0000

0.9999 0.0000 0.0000 0.0000

0.9999 0.0000 0.0000 0.0000

0.0000 0.0000 0.9999 0.0000

0.1429 0.6666 0.0953 0.0953

0.3333 0.2857 0.1429 0.2381

>MA0021.1_Dof3

0.9999 0.0000 0.0000 0.0000
```

3.3 Load files as motif list

We can load files using `read_motifs` function in `gimmemotifs`.

First, please prepare two files, `XXX.motif2factors.txt` and `XXX.pfm`. in the same directly. If you have theses two file in a different place, we cannot use the `read_motifs` function.

Then use file path for `XXX.pfm` for the argument of `read_motifs` function.

```
[58]: from gimmemotifs.motif import read_motifs

# Check path for pfm file
print(path)

# Read motifs
motifs = read_motifs(path)

# Check first 10 motifs
motifs[:10]
```

```
[58]: [MA0020.1_Dof2_AAAGCn,
MA0021.1_Dof3_AAAGyn,
MA0034.1_Gam1_nnyAACGmC,
MA0044.1_HMG-1_sTTGTnyTy,
MA0045.1_HMG-I/Y_nwAnAAAnrnmrAmAy,
MA0053.1_MNB1A_AAAGC,
MA0054.1_myb_Ph3_TAACnGTTw,
MA0064.1_PBF_AAAGY,
MA0082.1_squamosa_mC AwAwATrGwAAn,
MA0096.1_bZIP910_mtGACGT]
```

In another notebook, we introduce how to make `XXX.pfm` file and `XXX.motif2factors.txt` file. Pleas look at that if you want to make your motif data by yourself.

[Optional step 2] How to Make custom motifs for celloracle motif analysis

If you cannot find an appropriate motif dataset for your analysis and want to it by yourself. You can follow the instruction below. We introduce an example way to make motifs using [CisDB TF binding database](#).

0. Overview: How to Make custom motifs for celloracle motif scan.

In this notebook, we introduce how to make motif dataset for celloracle motif analysis. Celloracle uses list of motif object in `gimmemotifs` package. See `gimmemotifs` documentation for more details. (<https://gimmemotifs.readthedocs.io/en/master/api.html#>)

Here, we get motif data from CisBP (version2).<http://cisbp.ccbr.utoronto.ca>

We will extract motif information for a specific species and save as `XXX.pfm` and `XXX.motif2factors.txt` file. These files can be read with `read_motifs` function in `gimmemotifs`.

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
```

(continues on next page)

(continued from previous page)

```
import sys, os

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [20,10]

from gimmermotifs.motif import Motif, read_motifs
```

[]:

1. Download full dataset for TF dataset from CisBP database

Go the URL below for the download link for the latest data. <http://cisbp.ccbr.utoronto.ca/entireDownload.php>

```
[ ]: ! wget http://cisbp.ccbr.utoronto.ca/data/2.00/DataFiles/Bulk_downloads/EntireDataset/
    ↵PWMs.zip
! unzip PWMs.zip
```

```
[3]: !wget http://cisbp.ccbr.utoronto.ca/data/2.00/DataFiles/Bulk_downloads/EntireDataset/
    ↵TF_Information_all_motifs.txt.zip
! unzip TF_Information_all_motifs.txt.zip

--2020-08-03 16:34:06-- http://cisbp.ccbr.utoronto.ca/data/2.00/DataFiles/Bulk_
    ↵downloads/EntireDataset/TF_Information_all_motifs.txt.zip
Resolving cisbp.ccbr.utoronto.ca (cisbp.ccbr.utoronto.ca) ... 142.150.52.218
Connecting to cisbp.ccbr.utoronto.ca (cisbp.ccbr.utoronto.ca)|142.150.52.218|:80...
    ↵connected.
HTTP request sent, awaiting response... 200 OK
Length: 115455298 (110M) [application/zip]
Saving to: 'TF_Information_all_motifs.txt.zip'

TF_Information_all_ 100%[=====] 110.11M 11.1MB/s     in 19s

2020-08-03 16:34:25 (5.73 MB/s) - 'TF_Information_all_motifs.txt.zip' saved
    ↵[115455298/115455298]

Archive:  TF_Information_all_motifs.txt.zip
inflating: TF_Information_all_motifs.txt
```

```
[4]: # Load TF information as a dataframe.
df = pd.read_table("TF_Information_all_motifs.txt")
df.head()

/home/k/anaconda3/envs/pandas1/lib/python3.6/site-packages/IPython/core/
    ↵interactiveshell.py:3072: DtypeWarning: Columns (19) have mixed types.Specify dtype_
    ↵option on import or set low_memory=False.
    interactivity=interactivity, compiler=compiler, result=result)
```

	TF_ID	Family_ID	TSource_ID	Motif_ID	MSource_ID	DBID	\
0	T000001_2.00	F001_2.00	TS12_2.00	.	.	BRADI2G60554	
1	T000002_2.00	F001_2.00	TS12_2.00	.	.	LPERR05G06870	
2	T000003_2.00	F002_2.00	TS04_2.00	.	.	CPAG_02544	
3	T000004_2.00	F002_2.00	TS04_2.00	.	.	PTSG_00627	
4	T000005_2.00	F002_2.00	TS04_2.00	.	.	WUBG_06707	

(continues on next page)

(continued from previous page)

	TF_Name	TF_Species	TF_Status	Family_Name	...	\
0	BRADI2G60554	Brachypodium_distachyon	N	ABF1,B3	...	
1	LPERR05G06870	Leersia_perrieri	N	ABF1,B3	...	
2	CPAG_02544	Candida_parapsilosis	N	ABF1	...	
3	PTSG_00627	Salpingoeca_rosetta	N	ABF1	...	
4	WUBG_06707	Wuchereria_bancrofti	N	ABF1	...	
MSource_Year	PMID	MSource_Version	SR_Model	SR_NoThreshold	...	\
0	.	.	SequenceIdentity	True	...	
1	.	.	SequenceIdentity	True	...	
2	.	.	SequenceIdentity	True	...	
3	.	.	SequenceIdentity	True	...	
4	.	.	SequenceIdentity	True	...	
TfSource_Name		TfSource_URL	TfSource_Year	TfSource_Month	...	\
0	Ensembl	http://www.ensembl.org/	2018	Dec	...	
1	Ensembl	http://www.ensembl.org/	2018	Dec	...	
2	Broad	http://www.broadinstitute.org/	2016	May	...	
3	Broad	http://www.broadinstitute.org/	2016	May	...	
4	Broad	http://www.broadinstitute.org/	2016	May	...	
TfSource_Day						\
0	8					
1	8					
2	1					
3	1					
4	1					
 [5 rows x 28 columns]						

[14]: df.shape

(10879322, 28)

2. Define custom functions

[15]: # All process will be done inside these function.

```
from datetime import datetime
import glob

def read_pwn_and_convert_into_list(path):
    # read pwn as df
    pwm = pd.read_csv(path, delimiter="\t")

    # convert into list of str
    li = []
    for i in pwm.iterrows():
        i = i[1].values[1:]
        i = "\t".join(i.astype("str")) + "\n"
        li.append(i)

    return li

def make_motif_file_from_cisbp_data(pwm_folder_path, tfinfo_df, species):
```

(continues on next page)

(continued from previous page)

```

data_ = tfinfo_df[tfinfo_df.TF_Species == species]
data_name = "CisBP_ver2_" + species

## 1. Make file: motif2factors.txt

# Select information
df_factors = data_[["Motif_ID", "TF_Name", "MSource_Type", "TF_Status"]]
df_factors = df_factors[df_factors.TF_Status != "N"]

# Formatting
df_factors.columns = 'Motif\tFactor\tEvidence\tCurated'.split("\t")
df_factors["Curated"] = [{"D": "Y", "I": "N"}[i] for i in df_factors["Curated"]]
df_factors = df_factors.sort_values(by="Motif")

## 2. Make file: pfm file
comments = f"# CIS-BP motif database (v2.0), retrieved by Celloracle\n"
comments += "# Retrieved from: http://cisbp.ccb.utoronto.ca/data/2.00/DataFiles/\n"
comments += f"#Date: {datetime.now().ctime()}\n"

# Get list of motif name
paths_pwm = glob.glob(os.path.join(pwm_folder_path, "*.txt"))
paths_pwm.sort()
motif_names = [path.split("/")[-1].replace(".txt", "") for path in paths_pwm]
motifs = np.intersect1d(motif_names, df_factors.Motif.unique())

print(motifs.shape)

# Intersect motif information with pwm information
df_factors = df_factors[df_factors.Motif.isin(motifs)]

# Load, convert, and save pwm info
output = data_name + ".pfm"

motifs_non_zero = []
with open(output, "w") as f:

    for motif_name in motifs:

        path = os.path.join(pwm_folder_path, motif_name + ".txt")
        pwm = read_pwn_and_convert_into_list(path=path) # Load and convert
        if pwm:
            motifs_non_zero.append(motif_name)
            pwm = [f">{motif_name}\n"] + pwm
            for i in pwm: # Save pfm
                f.write(i)

# Intersect motif information with pwm information
df_factors = df_factors[df_factors.Motif.isin(motifs_non_zero)]

# Save factor info
df_factors.to_csv(f'{data_name}.motif2factors.txt', sep='\t', index=False)

print(df_factors.shape, len(motifs_non_zero))

```

3. Pick up motif information for one species and save as gmmemotif pfm file format.

```
[17]: # Check species in this dataset
species_list = df.TF_Species.unique()
species_list.sort()
```

```
for i in species_list:
    print(i)
```

```
Acanthamoeba_castellanii
Acanthamoeba_polyphaga_mimivirus
Acanthocheilonema_viteae
Acipenser_baerii
Acremonium_chrysogenum
Acropora_formosa
Acropora_millepora
Acyrthosiphon_pisum
Aedes_aegypti
Aegilops_tauschii
Agaricus_bisporus
Ailuropoda_melanoleuca
Albugo_laibachii
Alligator_sinensis
Allomyces_macrogyrus
Alternaria_brassicicola
Amanita_muscaria
Amborella_trichopoda
Amphimedon_queenslandica
Anas_platyrhynchos
Ancylostoma_caninum
Ancylostoma_ceylanicum
Ancylostoma_duodenale
Angiostrongylus_cantonensis
Angiostrongylus_costaricensis
Anisakis_simplex
Anncalilia_algerae
Anolis_carolinensis
Anopheles_albimanus
Anopheles_arabiensis
Anopheles_atroparvus
Anopheles_christyi
Anopheles_coluzzii
Anopheles_culicifacies
Anopheles_darlingi
Anopheles_dirus
Anopheles_epiroticus
Anopheles_farauti
Anopheles_funestus
Anopheles_gambiae
Anopheles_maculatus
Anopheles_melas
Anopheles_merus
Anopheles_minimus
Anopheles_quadriannulatus
Anopheles_sinensis
Anopheles_stephensi
Antirrhinum_majus
```

(continues on next page)

(continued from previous page)

Apis_mellifera
 Aplysia_californica
 Aptenodytes_forsteri
 Aquilegia_cerulea
 Arabidopsis_lyrata
 Arabidopsis_thaliana
 Artemia_franciscana
 Arthrobotrys_oligospora
 Arthroderma_benhamiae
 Arthroderma_otae
 Ascaris_lumbricoides
 Ascaris_suum
 Ashbya_gossypii
 Aspergillus_carbonarius
 Aspergillus_clavatus
 Aspergillus_flavus
 Aspergillus_fumigatus
 Aspergillus_nidulans
 Aspergillus_niger
 Aspergillus_oryzae
 Aspergillus_ruber
 Aspergillus_terreus
 Astyanax_mexicanus
 Atta_cephalotes
 Aureobasidium_melanogenum
 Aureobasidium_pullulans
 Aureobasidium_subglaciale
 Aureococcus_anophagefferens
 Avian_erythroblastosis_virus
 Avian_musculoaponeurotic_fibrosarcoma_virus_AS42
 Avian_myeloblastosis_virus
 Avian_sarcoma_virus_17
 Babesia_bovis
 Batrachochytrium_dendrobatidis
 Baudoinia_compniacensis
 Beauveria_bassiana
 Bigelowiella_natans
 Biomphalaria_glabrata
 Bipolaris_maydis
 Bipolaris_oryzae
 Bipolaris_sorokiniana
 Bipolaris_victoriae
 Bipolaris_zeicola
 Blastomyces_dermatitidis
 Blumeria_graminis
 Boechera_stricta
 Bombyx_mori
 Bos_grunniens
 Bos_taurus
 Botryobasidium_botryosum
 Botrytis_cinerea
 Bovine_papillomavirus_type_2
 Brachypodium_distachyon
 Branchiostoma_floridae
 Brassica_napus
 Brassica_oleracea
 Brassica_rapa

(continues on next page)

(continued from previous page)

Brettanomyces_bruxellensis
Brugia_malayi
Brugia_pahangi
Brugia_timori
Buceros_rhinoceros
Bursaphelenchus_xylophilus
Byssochlamys_spectabilis
Caenorhabditis_brenneri
Caenorhabditis briggsae
Caenorhabditis_elegans
Caenorhabditis_japonica
Caenorhabditis_remanei
Callithrix_jacchus
Calypte_anna
Camponotus_floridanus
Candida_albicans
Candida_dubliniensis
Candida_glabrata
Candida_guilliermondii
Candida_lusitaniae
Candida_maltosa
Candida_orthopsilosis
Candida_parapsilosis
Candida_tenuis
Candida_tropicalis
Canis_familiaris
Cannabis_sativa
Capitella_teleta
Capronia_coronata
Capronia_epimyces
Capronia_semiimmersa
Capsaspora_owczarzaki
Capsella_grandiflora
Capsella_rubella
Carica_papaya
Cavia_porcellus
Ceriporiopsis_subvermispora
Chaetomium_globosum
Chaetomium_thermophilum
Chelonia_mydas
Chlamydomonas_reinhardtii
Chlorella_NC64A
Chlorella_vulgaris
Chlorocebus_sabaeus
Choloepus_hoffmanni
Chroomonas_mesostigmatica
Cicer_arietinum
Cimex_lectularius
Ciona_intestinalis
Ciona_savignyi
Citrullus_lanatus
Citrus_clementina
Citrus_sinensis
Cladophialophora_bantiana
Cladophialophora_carrionii
Cladophialophora_immunda
Cladophialophora_psammophila

(continues on next page)

(continued from previous page)

Cladophialophora_yegresii
 Claviceps_purpurea
 Clavispora_lusitaniae
 Clonorchis_sinensis
 Coccidioides_immitis
 Coccidioides_posadasii
 Coccomyxa_subellipsoidea_C169
 Cochliobolus_heterostrophus_C5
 Colletotrichum_fioriniae
 Colletotrichum_gloeosporioides
 Colletotrichum_graminicola
 Colletotrichum_higginsianum
 Colletotrichum_orbiculare
 Colletotrichum_sublineola
 Coniosporium_apollinis
 Coprinellus_disseminatus
 Coprinopsis_cinerea
 Coprinopsis_scobicola
 Cordyceps_militaris
 Crassostrea_gigas
 Cryphonectria_parasitica
 Cryptococcus_gattii
 Cryptococcus_neoformans
 Cryptosporidium_hominis
 Cryptosporidium_muris
 Cryptosporidium_parvum
 Cucumis_sativus
 Culex_pipiens
 Culex_quinquefasciatus
 Cupiennius_salei
 Cyanidioschyzon_merolae
 Cylicostephanus_goldi
 Cyphelophora_europaea
 Dacryopinax_sp
 Dactylellina_haptotyla
 Danaus_plexippus
 Danio_rerio
 Daphnia_pulex
 Dasypus_novemcinctus
 Debaryomyces_hansenii
 Dendroctonus_ponderosae
 Dichomitus_squalens
 Dictyocaulus_viviparus
 Dictyostelium_discoideum
 Dictyostelium_purpureum
 Diphyllothrix_latum
 Dipodomys_ordii
 Dirofilaria_immitis
 Discocelis_tigrina
 Dothistroma_septosporum
 Dracunculus_medinensis
 Drechslerella_stenobrocha
 Drosophila_ananassae
 Drosophila_erecta
 Drosophila_grimshawi
 Drosophila_melanogaster
 Drosophila_mojavensis

(continues on next page)

(continued from previous page)

Drosophila_persimilis
Drosophila_pseudoobscura
Drosophila_sechellia
Drosophila_simulans
Drosophila_virilis
Drosophila_willistoni
Drosophila_yakuba
Echinococcus_canadensis
Echinococcus_granulosus
Echinococcus_multilocularis
Echinops_telfairi
Echinostoma_caproni
Edhazardia_aedis
Eimeria_tenella
Elaeophora_elaphi
Eleutheria_dichotoma
Emiliania_huxleyi
Encephalitozoon_cuniculi
Encephalitozoon_hellem
Encephalitozoon_intestinalis
Encephalitozoon_romaleae
Endocarpon_pusillum
Entamoeba_dispar
Entamoeba_histolytica
Entamoeba_invadens
Enterobius_vermicularis
Enterocytozoon.bieneusi
EpsteinBarr_virus
Equus_caballus
Eremothecium_cymbalariae
Erinaceus_europaeus
Erysiphe_necator
Eucalyptus_grandis
Eutrema_salsugineum
Eutypa_lata
Exophiala_aquamarina
Exophiala_dermatitidis
Exophiala_mesophila
Exophiala_oligosperma
Exophiala_sideris
Exophiala_spinifera
Exophiala_xenobiotica
Fasciola_hepatica
Felis_catus
Fibroporia_radiculosa
Ficedula_albicollis
Fomitopsis_pinicola
Fonsecaea_pedrosoi
Fragaria Vesca
Fragilariopsis_cylindrus
Fusarium_fujikuroi
Fusarium_graminearum
Fusarium_oxysporum
Fusarium_pseudograminearum
Fusarium_solani
Fusarium_verticillioides
Gadus_morhua

(continues on next page)

(continued from previous page)

Gaeumannomyces_graminis
 Galerina_marginata
 Gallid_herpesvirus_2
 Gallus_gallus
 Gasterosteus_aculeatus
 Geospiza_fortis
 Giardia_lamblia
 Glarea_lozoyensis
 Globodera_pallida
 Gloeophyllum_trabeum
 Glycine_max
 Gongylonema_pulchrum
 Gorilla_gorilla
 Gossypium_raimondii
 Grosmannia_clavigera
 Guillardia_theta
 Gymnopus_luxurians
 Haemonchus_contortus
 Haemonchus_placei
 Halocynthia_roretzi
 Halyomorpha_halys
 Harpegnathos_saltator
 Hebeloma_cylindrosporum
 Helianthus_annuus
 Heliconius_melpomene
 Heligmosomoides_bakeri
 Helobdella_robusta
 Hemiselmis_andersenii
 Heterobasidion_annosum
 Heterobasidion_irregulare
 Heterodontus_francisci
 Heterorhabditis_bacteriophora
 Histoplasma_capsulatum
 Homo_sapiens
 Hordeum_vulgare
 Hyaloperonospora_arabidopsis
 Hydatigera_taeniaeformis
 Hydnomerulius_pinastri
 Hydra_magnipapillata
 Hymenolepis_diminuta
 Hymenolepis_microstoma
 Hymenolepis_nana
 Ictidomys_tridecemlineatus
 Ipomoea_batatas
 Ixodes_scapularis
 Jaapia_argillacea
 Kazachstanica_africana
 Kazachstanica_naganishii
 Kluyveromyces_delphensis
 Kluyveromyces_lactis
 Kluyveromyces_thermotolerans
 Kluyveromyces_waltii
 Komagataella_pastoris
 Kuraishia_capsulata
 Laccaria_amethystina
 Laccaria_bicolor
 Lachancea_kluyveri

(continues on next page)

(continued from previous page)

Lachancea_lanzarotensis
Lachancea_thermotolerans
Latimeria_chalumnae
Leersia_perrieri
Leishmania_braziliensis
Leishmania_infantum
Leishmania_major
Leishmania_mexicana
Lepisosteus_oculatus
Leptinotarsa_decemlineata
Leptosphaeria_maculans
Lingula_unguis
Linum_usitatissimum
Litomosoides_sigmodontis
Loa_loa
Lodderomyces_elongisporus
Lottia_gigantea
Lotus_japonicus
Loxodonta_africana
Lucilia_cuprina
Lutzomyia_longipalpis
Lycopersicon_esculentum
Macaca_fascicularis
Macaca_mulatta
Macrohomina_phaseolina
Macropus_eugenii
Magnaporthe_oryzae
Magnaporthe_poae
Malassezia_globosa
Malassezia_sympodialis
Malus_domestica
Manihot_esculenta
Marmota_monax
Marssonina_brunnea
Medicago_sativa
Medicago_truncatula
Megaselia_scalaris
Melampsora_laricipopulina
Meleagris_gallopavo
Melitaea_cinxia
Meloidogyne_floridensis
Meloidogyne_hapla
Meloidogyne_incognita
Mesocestoides_corti
Metarhizium_acridum
Metarhizium_album
Metarhizium_anisopliae
Metarhizium_brunneum
Metarhizium_guizhouense
Metarhizium_majus
Metarhizium_robertsii
Meyerozyma_guilliermondii
Microbotryum_violaceum
Microcebus_murinus
Micromonas_pusilla
Micromonas_sp_RCC299
Microsporidia_sp

(continues on next page)

(continued from previous page)

Microsporum_canis
Microsporum_gypseum
Millerozyma_farinosa
Mimulus_guttatus
Mixia_osmundae
Mnemiopsis_leidyi
Moniliophthora_perniciosa
Moniliophthora_roreri
Monodelphis_domestica
Monosiga_brevicollis
Mucor_circinelloides
Mus_musculus
Musa_acuminata
Musca_domestica
Mustela_putorius_furo
Myceliophthora_thermophila
Mycosphaerella_fijiensis
Mycosphaerella_graminicola
Myotis_brandtii
Myotis_lucifugus
Naegleria_gruberi
Nasonia_vitripennis
Naumovozyma_castellii
Naumovozyma_dairenensis
Necator_americanus
Nectria_haematoxocca
Nematocida_parisi_ertml
Nematocida_sp
Nematostella_vectensis
Neofusicoccum_parvum
Neosartorya_fischeri
Neospora_caninum
Neurospora_crassa
Neurospora_discreta
Neurospora_tetrasperma
Nicotiana_sp.
Nicotiana_tabacum
Nippostrongylus_brasiliensis
Nomascus_leucogenys
Nosema_apis
Nosema_bombycis
Nosema_ceranae
Ochotona_princeps
Octopus_bimaculoides
Oesophagostomum_dentatum
Ogataea_parapolymorpha
Oidiodendron_maius
Oikopleura_dioica
Onchocerca_flexuosa
Onchocerca_ochengi
Onchocerca_volvulus
Oncorhynchus_tshawytscha
Ophiocordyceps_sinensis
Ophiostoma_piceae_uamh
Ophisaurus_gracilis
Opisthorchis_viverrini
Ordospora_colligata

(continues on next page)

(continued from previous page)

Oreochromis_niloticus
Ornithorhynchus_anatinus
Oryctolagus_cuniculus
Oryza_barthii
Oryza_brachyantha
Oryza_glaberrima
Oryza_glumaepatula
Oryza_indica
Oryza_longistaminata
Oryza_meridionalis
Oryza_nivara
Oryza_punctata
Oryza_rufipogon
Oryza_sativa
Oryzias_latipes
Ostreococcus_RCC809
Ostreococcus_lucimarinus
Ostreococcus_tauri
Otolemur_garnettii
Ovis_aries
Oxytricha_trifallax
PBM_CONSTRUCTS
Pan_paniscus
Pan_troglodytes
Panicum_virgatum
Papio_anubis
Paracoccidioides_brasiliensis
Paracoccidioides_sp_lutzii
Paramecium_tetraurelia
Parascaris_equorum
Parastrongyloides_trichosuri
Patiria_miniata
Paxillus_involutus
Paxillus_rubicundulus
Pediculus_humanus
Pelodiscus_sinensis
Penicillium_chrysogenum
Penicillium_digitatum
Penicillium_expansum
Penicillium_italicum
Penicillium_marneffei
Penicillium_oxalicum
Penicillium_rubens
Penicillium_solitum
Pestalotiopsis_fici
Petromyzon_marinus
Petroselinum_crispum
Petunia_x_hybrid
Phaeodactylum_tricornutum
Phaeosphaeria_nodorum
Phanerochaete_carnosa
Phanerochaete_chrysosporium
Phaseolus_vulgaris
Phlebiopsis_gigantea
Phlebotomus_papatasi
Phoenix_dactylifera
Phycomyces_blakesleeanus

(continues on next page)

(continued from previous page)

Physcomitrella_patens
Phytophthora_capsici
Phytophthora_infestans
Phytophthora_kernoviae
Phytophthora_lateralis
Phytophthora_parasitica
Phytophthora_ramorum
Phytophthora_sojae
Pichia_angusta
Pichia_kudriavzevii
Pichia_pastoris
Pichia_stipitis
Piloderma_croceum
Piriformospora_indica
Pisolithus_microcarpus
Pisolithus_tinctorius
Pisum_sativum
Plasmodium_berghei
Plasmodium_chabaudi
Plasmodium_falciparum
Plasmodium_knowlesi
Plasmodium_vivax
Plasmodium_yoelii
Pleurobrachia_pileus
Pleurotus_djamor
Pleurotus_ostreatus
Plicaturopsis_crispa
Pneumocystis_carinii
Pneumocystis_murina
Podocoryne_carnea
Podospora_anserina
Poecilia_formosa
Pogona_vitticeps
Polysphondylillum_pallidum
Pongo Abelii
Populus_trichocarpa
Postia_placenta
Pristionchus_exspectatus
Pristionchus_pacificus
Procavia_capensis
Protopolystoma_xenopodis
Prunus_mume
Prunus_persica
Pseudocercospora_fijiensis
Pseudogymnoascus_destructans
Pseudogymnoascus_pannorum
Pseudozyma_antarctica
Pseudozyma_aphidis
Pseudozyma_brasiliensis
Pseudozyma_flocculosa
Pseudozyma_hubeiensis
Pteropus_vampyrus
Puccinia_graminis
Puccinia_triticina
Punctularia_strigosozonata
Pyrenophora_teres
Pyrenophora_triticirepentis

(continues on next page)

(continued from previous page)

Pythium_aphanidermatum
Pythium_arrenomanes
Pythium_irregulare
Pythium_iwayamai
Pythium_ultimum
Pythium_vexans
Rattus_norvegicus
Rhabditophanes_kr3021
Rhinocladiella_mackenziei
Rhizoctonia_solani
Rhizophagus_irregularis
Rhizopus_oryzae
Rhodnius_prolixus
Rhodosporidium_toruloides
Rhodotorula_glutinis
Ricinus_communis
Romanomermis_culicivorax
Rozella_allomycis
Saccharomyces_arboricola
Saccharomyces_bayanus
Saccharomyces_castellii
Saccharomyces_cerevisiae
Saccharomyces_kudriavzevii
Saccharomyces_mikatae
Saccharomyces_paradoxus
Saccharomyctaceae_sp_ashbya_aceri
Saccoglossus_kowalevskii
Salix_purpurea
Salpingoeca_rosetta
Saprolegnia_parasitica
Sarcophilus_harrisii
Sarsia_sp._Long_Island_Sound
Scedosporium_apiospermum
Scheffersomyces_stipitis
Schistocephalus_solidus
Schistosoma_curassoni
Schistosoma_haematobium
Schistosoma_japonicum
Schistosoma_mansoni
Schistosoma_margrebowiei
Schistosoma_mattheei
Schistosoma_rodhaini
Schizophyllum_commune
Schizosaccharomyces_cryophilus
Schizosaccharomyces_japonicus
Schizosaccharomyces_octosporus
Schizosaccharomyces_pombe
Schmidtea_mediterranea
Scleroderma_citrinum
Sclerotinia_borealis
Sclerotinia_sclerotiorum
Selaginella_moellendorffii
Serendipita_vermifera
Serpula_lacrymans
Setaria_italica
Setosphaeria_turcica
Soboliphyme_baturini

(continues on next page)

(continued from previous page)

Solanum_lycopersicum
 Solanum_tuberosum
 Solenopsis_invicta
 Sordaria_macrospora
 Sorex_araneus
 Sorghum_bicolor
 Spadella_cephaloptera
 Spathaspora_passalidarum
 Sphaerobolus_stellatus
 Sphaerulina_musiva
 Spirometra_erinaceieuropaei
 Spizellomyces_punctatus
 Sporisorium_reilianum
 Sporobolomyces_roseus
 Sporothrix_brasiliensis
 Sporothrix_schenckii
 Spraguea_lophii
 Stachybotrys_chartarum
 Stachybotrys_chlorohalonata
 Stagonospora_nodorum
 Steinernema_carpocapsae
 Steinernema_feltiae
 Steinernema_glaseri
 Steinernema_monticolum
 Steinernema_scapterisci
 Strigamia_maritima
 Strongylocentrotus_purpuratus
 Strongyloides_papillosus
 Strongyloides_stercoralis
 Strongyloides_venezuelensis
 Strongylus_vulgaris
 Suillus_luteus
 Sus_scrofa
 Syphacia_muris
 Taenia_asiatica
 Taenia_solium
 Taeniopygia_guttata
 Takifugu_rubripes
 Talaromyces_marneffei
 Talaromyces_stipitatus
 Tarsius_syrichta
 Teladorsagia_circumcincta
 Tetrahymena_thermophila
 Tetranychus_urticae
 Tetraodon_nigroviridis
 Tetrapisispora_blaetae
 Tetrapisispora_phaffii
 Thalassiosira_pseudonana
 Thecamonas_trahens
 Theileria_annulata
 Theileria_parva
 Thelazia_callipaeda
 Theobroma_cacao
 Thielavia_terrestris
 Tilletiaria_anomala
 Togninia_minima
 Torrubiella_hemipterigena

(continues on next page)

(continued from previous page)

Torulaspora_delbrueckii
Toxocara_canis
Toxoplasma_gondii
Trachipleistophora_hominis
Trametes_cinnabarina
Tremella_mesenterica
Tribolium_castaneum
Trichinella_nativa
Trichinella_spiralis
Trichobilharzia_regenti
Trichoderma_atroviride
Trichoderma_reesei
Trichoderma_virens
Trichomonas_vaginalis
Trichophyton_equinum
Trichophyton_interdigitale
Trichophyton_rubrum
Trichophyton_soudanense
Trichophyton_tonsurans
Trichophyton_verrucosum
Trichoplax_adhaerens
Trichosporon_asahii
Trichuris_muris
Trichuris_suis
Trichuris_trichiura
Trionyx_sinensis
Tripedalia_cystophora
Triticum_aestivum
Triticum_urartu
Trypanosoma_brucei
Trypanosoma_congolense
Trypanosoma_cruzi
Trypanosoma_vivax
Tuber_melanosporum
Tulasnella_calospora
Tupaia_belangeri
Tursiops_truncatus
Tyto_alba
Uncinocarpus_reesii
Ustilaginoidea_virens
Ustilago_hordei
Ustilago_maydis
Vanderwaltozyma_polyspora
Vavraia_culicis
Verruconis_gallopava
Verticillium_albo_atrum
Verticillium_alfalfae
Verticillium_dahliae
Vicugna_pacos
Vitis_vinifera
Vittaforma_corneae
Volvox_carteri
Wallemia_ichthyophaga
Wallemia_sebi
Wickerhamomyces_ciferrii
Wuchereria_bancrofti
Xenopus_laevis

(continues on next page)

(continued from previous page)

```
Xenopus_tropicalis
Xiphophorus_maculatus
Yarrowia_lipolytica
Zea_mays
Zootermopsis_nevadensis
Zygosaccharomyces_bailii
Zygosaccharomyces_rouxii
Zymoseptoria_tritici
```

```
[18]: # Pick up motif information for one species and save as gimmelmotif pfm file format.

species = "Danio_rerio"
make_motif_file_from_cisbp_data(pwm_folder_path="pwms", tfinfo_df=df, species=species)
(6133,)
(109560, 4) 5298
```

4. Check results

```
[19]: # Read motifs

from gimmelmotifs.motif import read_motifs

path = f"CisBP_ver2_{species}.pfm"
custom_motifs = read_motifs(path)
custom_motifs[:10]
```

```
[19]: [M00008_2.00_nnnAAww,
M00045_2.00_GTAAACAA,
M00056_2.00_TAATAAAT,
M00066_2.00_nsGTTGCyAn,
M00070_2.00_nrAACAAATAnn,
M00111_2.00_nGCCynnGGs,
M00112_2.00_CCTsrGGCnA,
M00113_2.00_nsCCnnAGGs,
M00114_2.00_nnGCCynnGG,
M00115_2.00_nnATnAAAn]
```

```
[ ]: # Delete downloaded data
```

```
[1]: ! rm -r TF_Information_all_motifs*
! rm PWMs.*
! rm -r pwms
```

1.2.3 Single-cell RNA-seq data preprocessing

Network analysis and simulation in celloracle will be performed using scRNA-seq data. The scRNA-seq data should include the components below.

- Gene expression matrix; mRNA counts before scaling and transformation.
- Clustering results.
- Dimensional reduction results.

In addition to these minimum requirements, we highly recommend doing these analyses below in the preprocessing step.

- Data quality check and cell/gene filtering.
- Normalization
- Identification of highly variable genes

We recommend processing scRNA-seq data using either Scanpy or Seurat. If you are not familiar with the general workflow of scRNA-seq data processing, please go to [the documentation for scanpy](#) and [the documentation for Seurat](#) before celloracle analysis.

If you already have preprocessed scRNA-seq data, which includes the necessary information above, you can skip this part.

A. scRNA-seq data preprocessing with scanpy

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

In this tutorial, we introduce an example of scRNA-seq preprocessing for celloracle with scanpy. We wrote the notebook based on [one of scanpy's tutorials](#) with some modifications.

The jupyter notebook files and data used in this tutorial are available [here](#).

Python notebook

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"] = [6, 4.5]
```

1. Load data

In this notebook, we will show an example of how to process scRNA-seq data using a 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>). You can easily download this scRNA-seq data with a scanpy 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

```
[4]: # Only consider genes with more than 1 count
sc.pp.filter_genes(adata, min_counts=1)
```

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. Identification of highly variable genes

Removing non-variable genes not only reduces the calculation time during the GRN reconstruction and simulation, but also improve the accuracy of GRN inference. We recommend using the top 2000~3000 variable genes.

```
[6]: # Select top 2000 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

We will do log transformation scaling because these are necessary for PCA, clustering, and differential gene calculations. However, we also need non-transformed gene expression data in the celloracle analysis. Thus we keep raw count in anndata using the following command before the log transformation.

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

6. Dimensional reduction

Dimensional reduction is one of the most important parts of the scRNA-seq analysis. Celloracle needs dimensional reduction embeddings to simulate cell transition.

Please choose a proper algorithm for dimensional reduction so that the embedding appropriately represents the data structure. We recommend using one of these dimensional reduction algorithms (or trajectory inference algorithms); UMAP, tSNE, diffusion map, force-directed graph drawing or PAGA.

In this example, we use a combination of four algorithms; diffusion map, 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 neighbors again based on diffusionmap  
sc.pp.neighbors(adata, n_neighbors=10, use_rep='X_diffmap')
```

7. Clustering

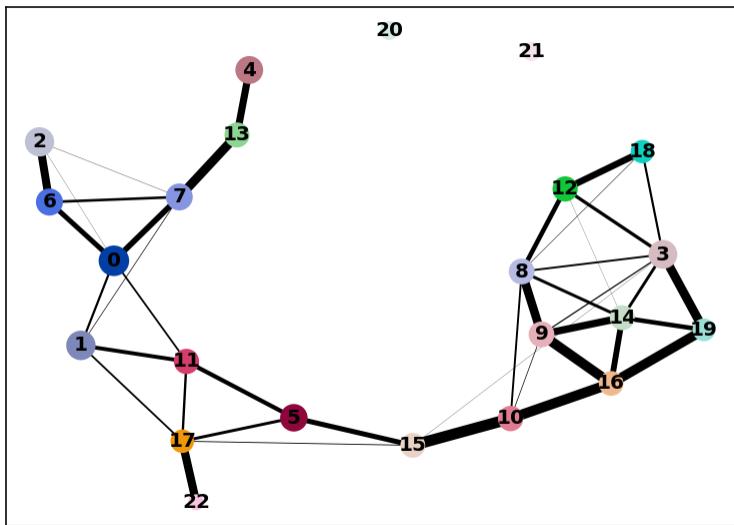
```
[11]: sc.tl.louvain(adata, resolution=0.8)
```

(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]
```

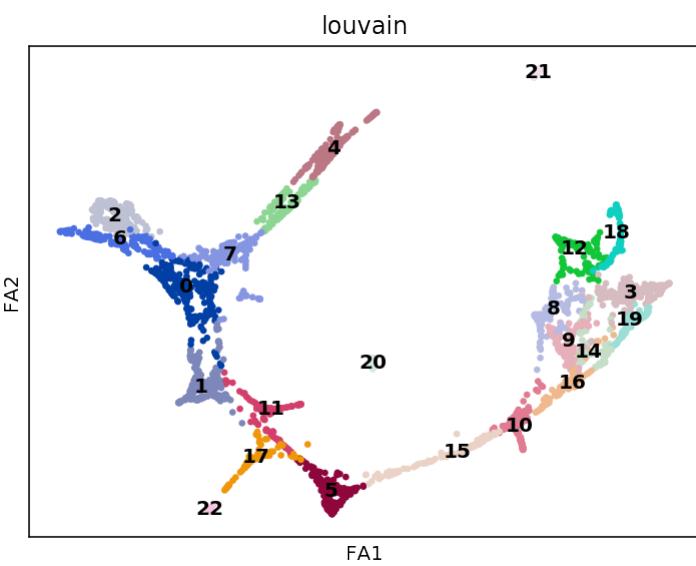
```
[14]: plt.rcParams["figure.figsize"] = [6, 4.5]
```

```
[15]: sc.pl.paga(adata)
```



```
[16]: sc.tl.draw_graph(adata, init_pos='paga', random_state=123)
```

```
[17]: sc.pl.draw_graph(adata, color='louvain', legend_loc='on data')
```



8. Check data

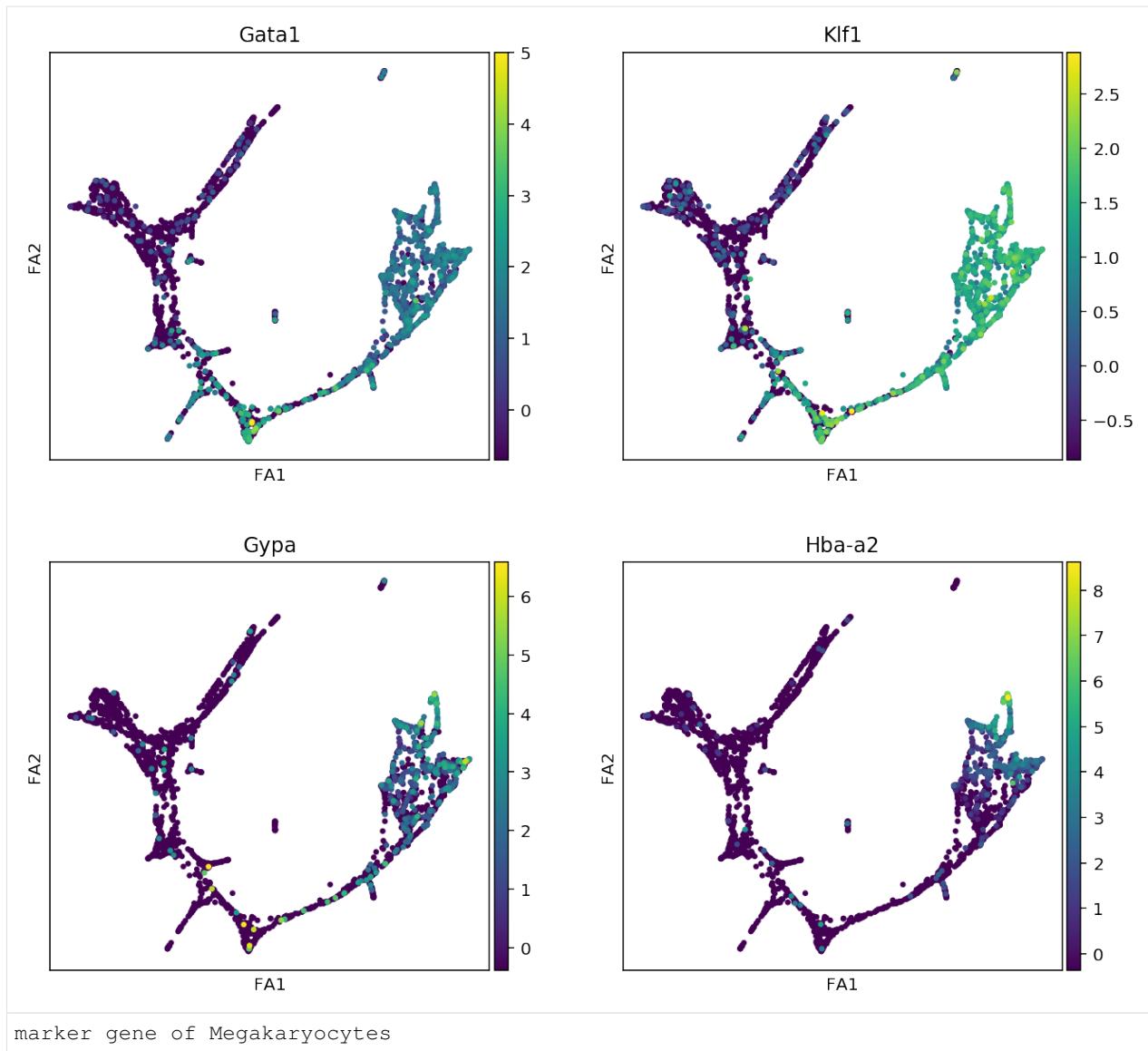
8.1. Visualize marker gene expression

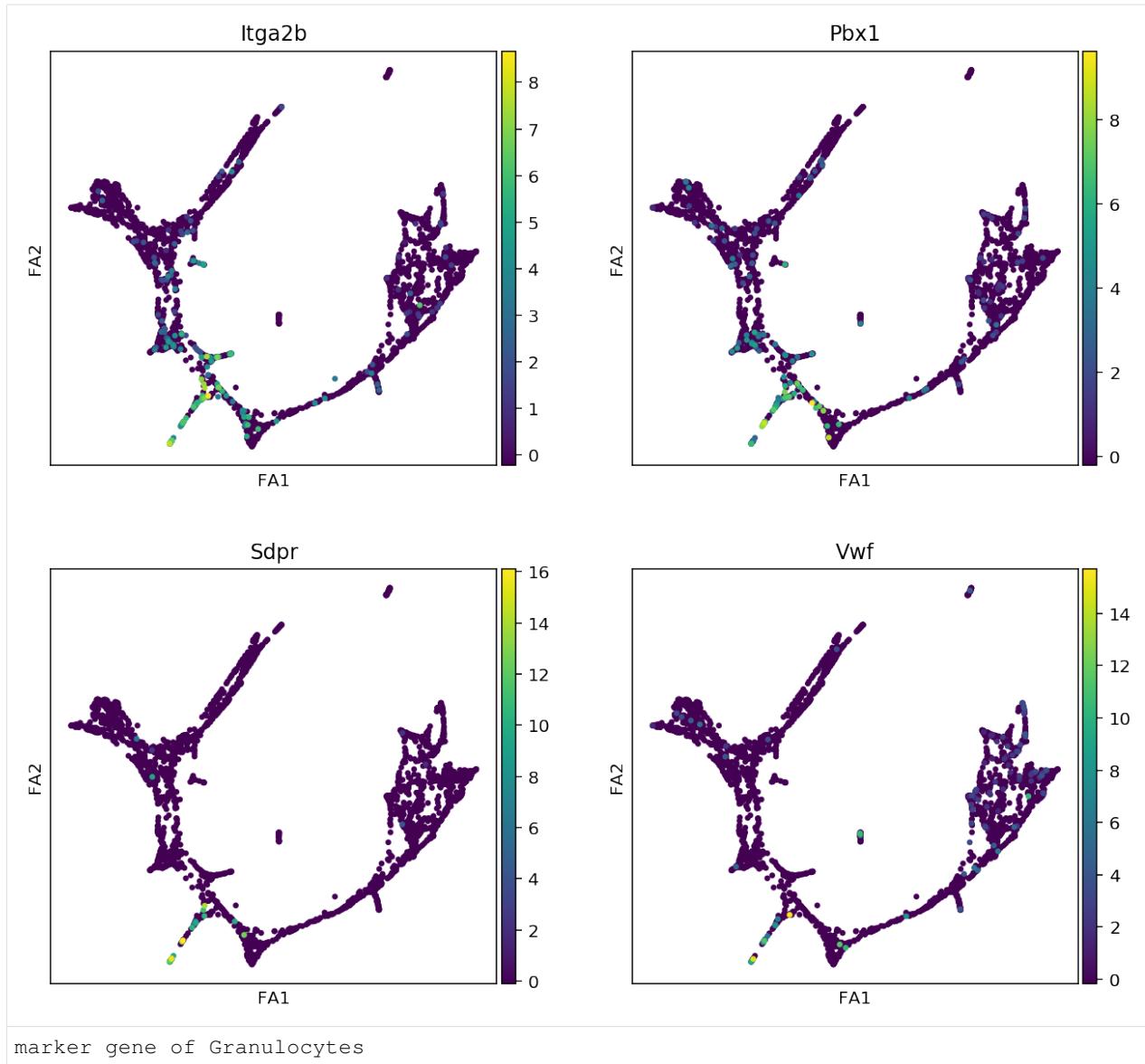
```
[18]: plt.rcParams["figure.figsize"] = [4.5, 4.5]

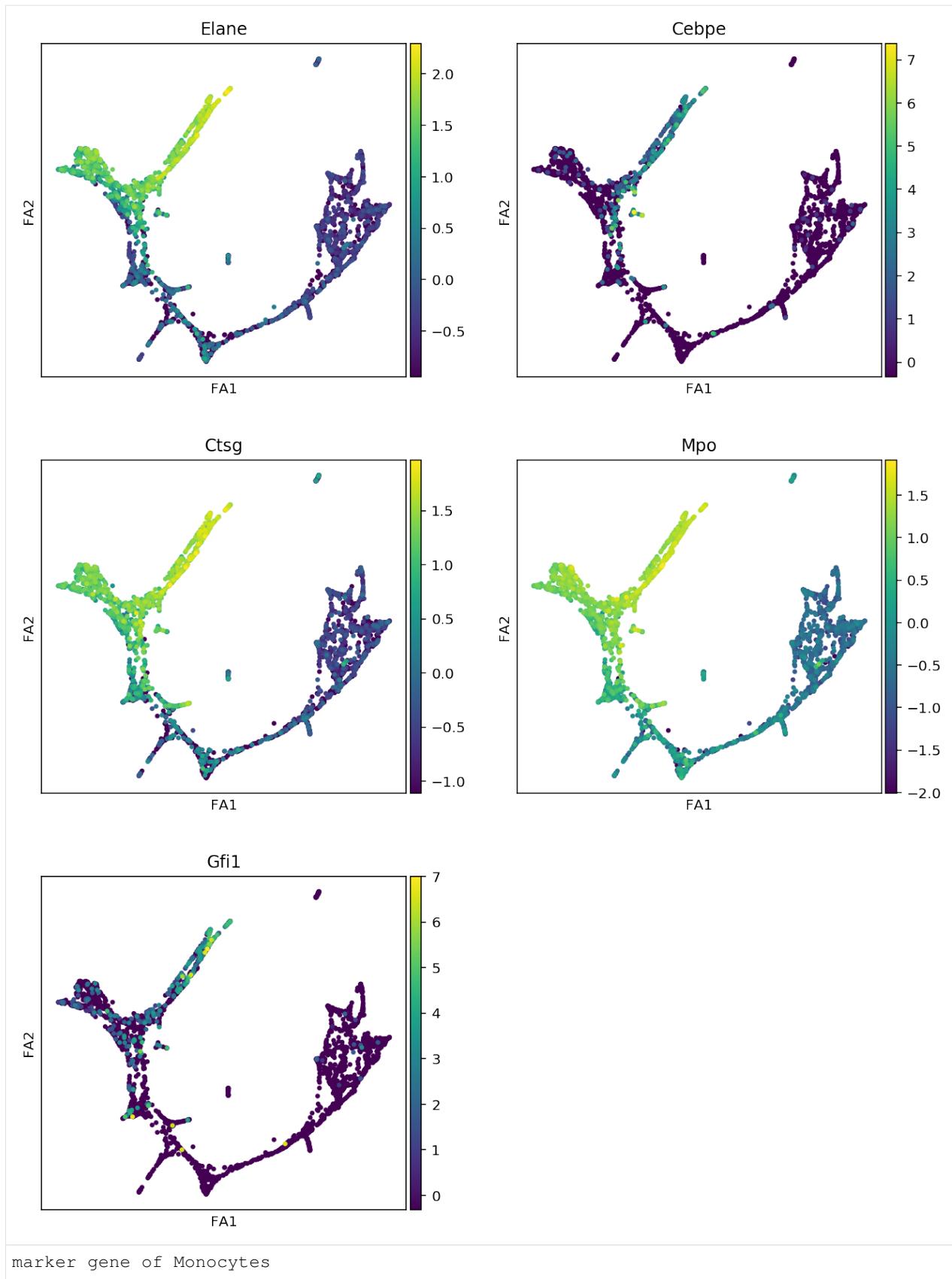
[19]: markers = {"Erythroids": ["Gata1", "Klf1", "Gypa", "Hba-a2"],
   "Megakaryocytes": ["Itga2b", "Pbx1", "Sdpr", "Vwf"],
   "Granulocytes": ["Elane", "Cebpe", "Ctsg", "Mpo", "Gfil"],
   "Monocytes": ["Irf8", "Csflr", "Ctsg", "Mpo"],
   "Mast_cells": ["Cma1", "Gzmb", "Kit"],
   "Basophils": ["Mcpt8", "Prss34"]}
}

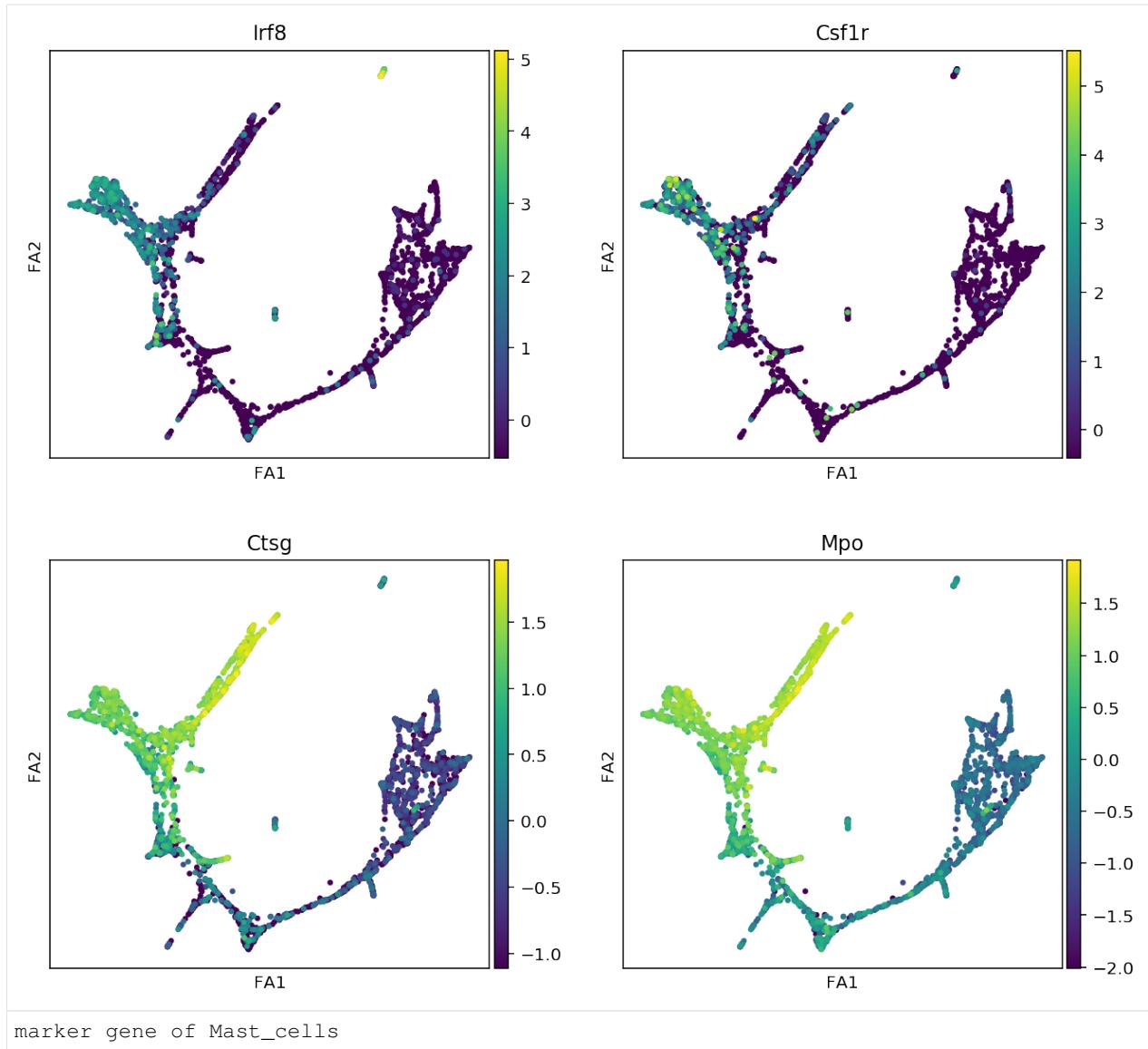
for cell_type, genes in markers.items():
    print(f"marker gene of {cell_type}")
    sc.pl.draw_graph(adata, color=genes, use_raw=False, ncols=2)
    plt.show()
```

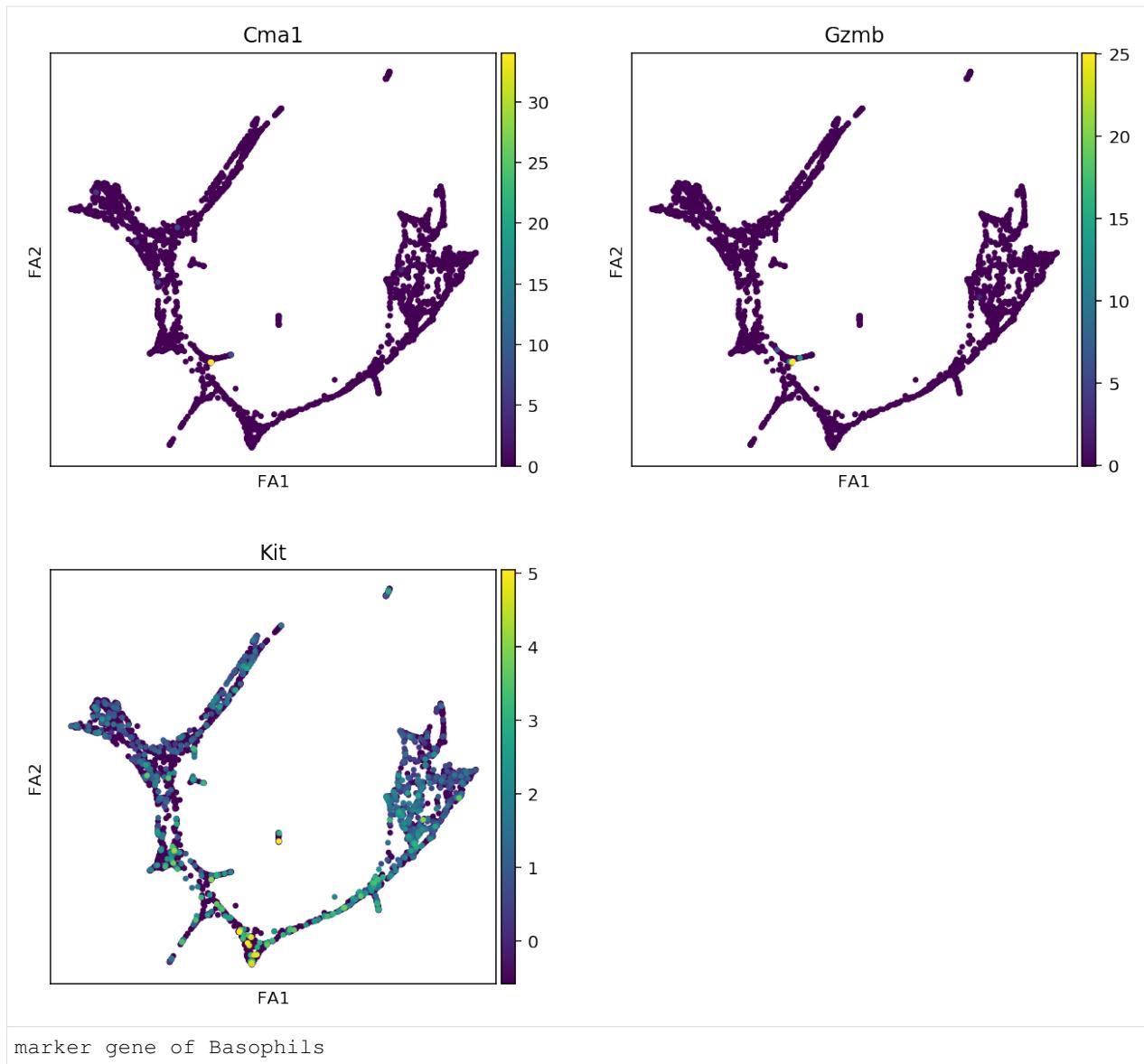
```
marker gene of Erythroids
```

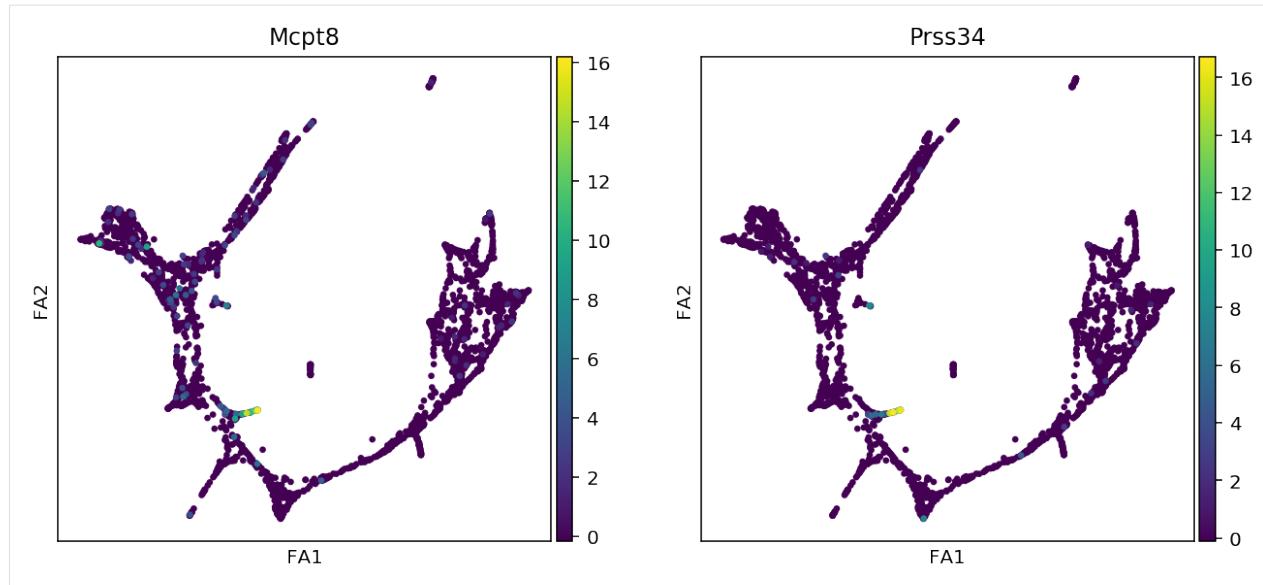










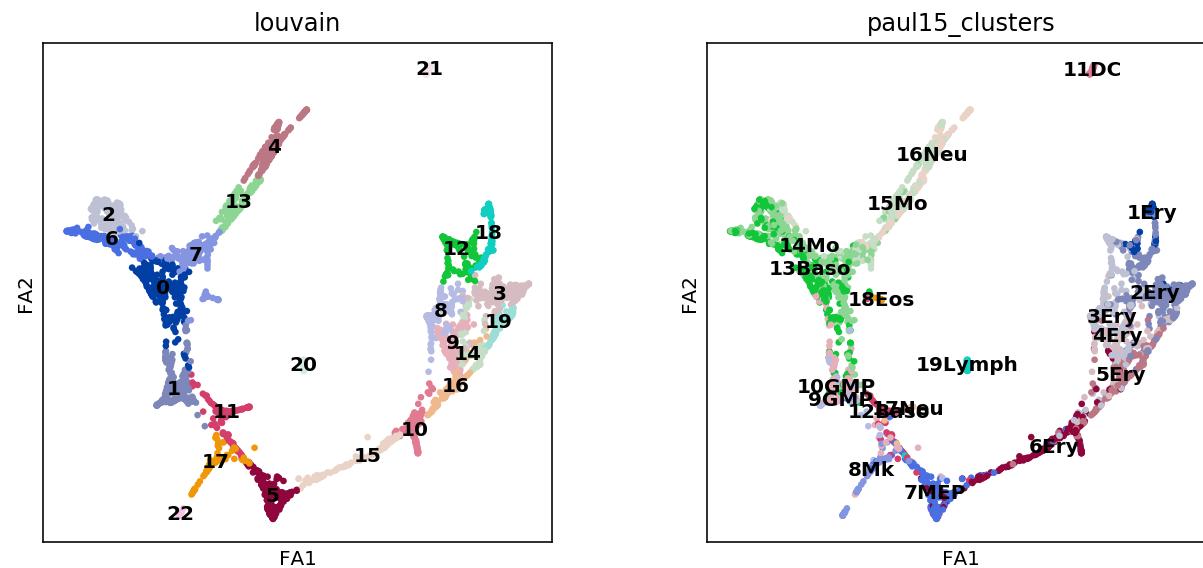


9. Make annotation for cluster

Based on the marker gene expression and previous reports, we will manually annotate each cluster. When using your own data, you will need to annotate the clusters appropriately.

9.1. Make annotation (1)

```
[20]: sc.pl.draw_graph(adata, color=['louvain', 'paul15_clusters'],
                      legend_loc='on data')
```



```
[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]
```

!! Please change the dictionary below depending on the clustering results. The results may change depending on the execution environment.

```
[22]: # Make annotation 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]}

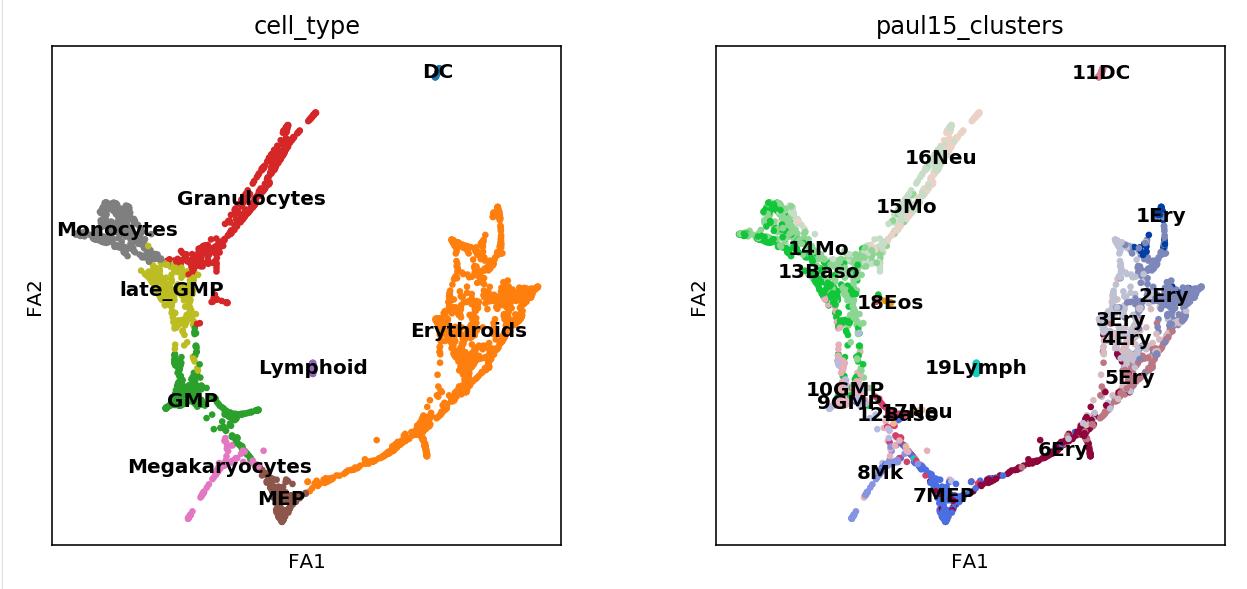
# change dictionary format
annotation_rev = {}
for i in cluster_list:
    for k in annotation:
        if int(i) in annotation[k]:
            annotation_rev[i] = k

# 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]
```

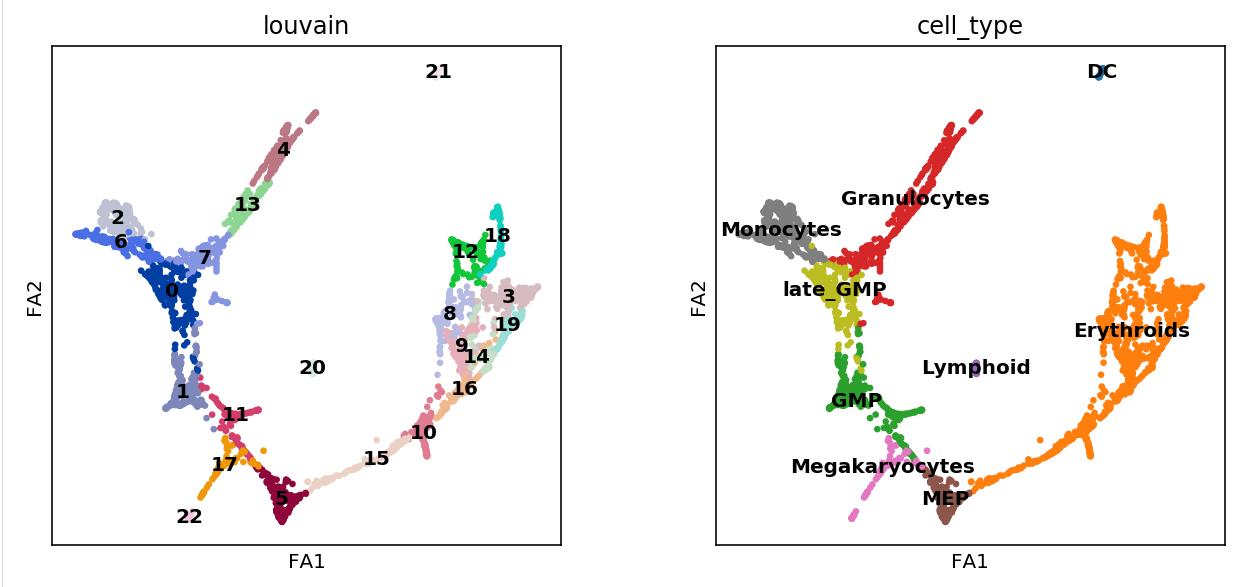
```
[24]: # check results
sc.pl.draw_graph(adata, color=['cell_type', 'paul15_clusters'],
                 legend_loc='on data')
... storing 'cell_type' as categorical
```



9.2. 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')
```

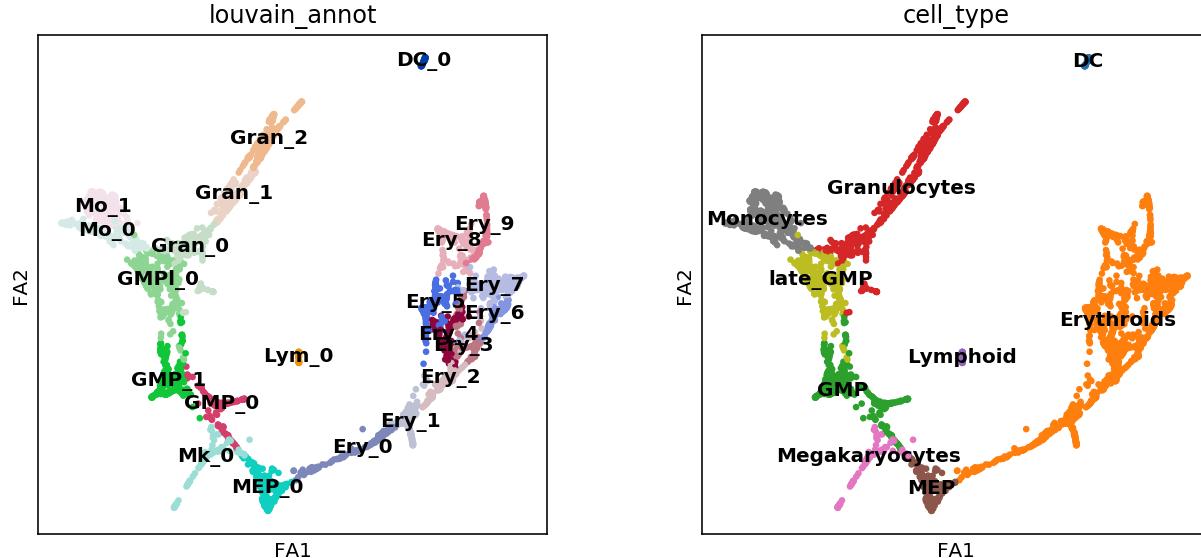


!! Please change the dictionary below depending on the clustering results. The results may change depending on the execution environment.

```
[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': 'GMP1_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
```



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

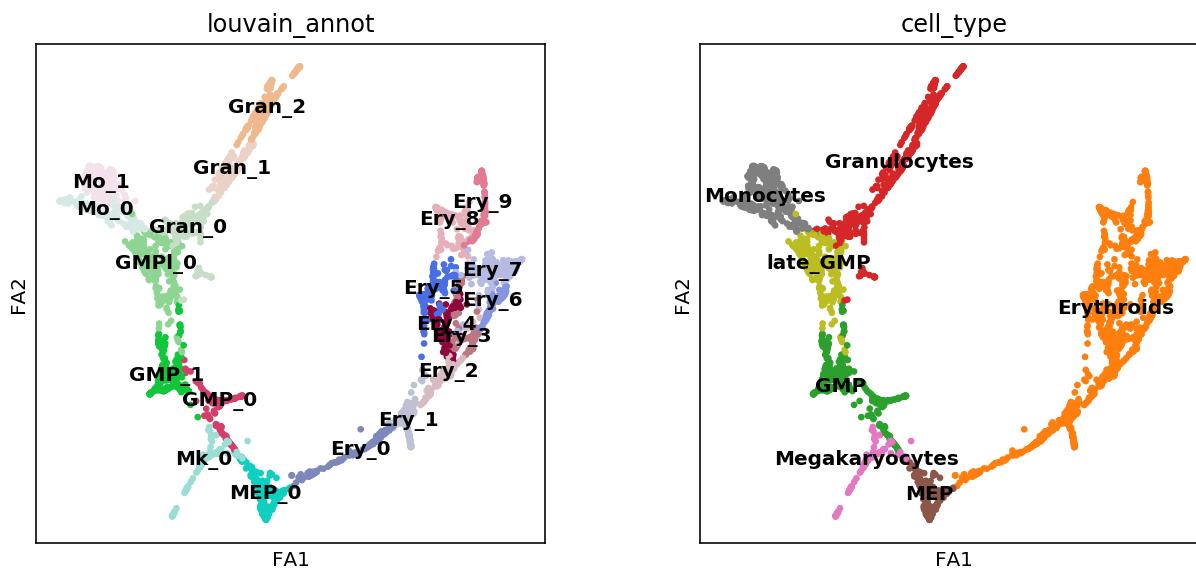
10. (Option) Subset cells

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 will remove non-myeloid cell cluster; DC and Lymphoid cell cluster.

```
[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')
```



11. Save data

```
[32]: adata.write_h5ad("data/Paul_et al_15.h5ad")
```

B. scRNA-seq data preprocessing with Seurat

R notebook ... comming in the future update.

Note: If you use Seurat for preprocessing, you need to convert the scRNA-seq data (Seurat object) into anndata to analyze the data with celloracle. celloracle has a python API and command-line API to convert a Seurat object into an anndata. Please go to the documentation of celloracle's API documentation for more information.

1.2.4 Network analysis

celloracle imports the scRNA-seq dataset and TF binding information to find active regulatory connections for all genes, generating sample-specific GRNs.

The inferred GRN is analyzed with several network algorithms to get various network scores. The network score is useful to identify key regulatory genes.

Celloracle reconstructs a GRN for each cluster, enabling us to compare GRNs to each other. It is also possible to analyze how the GRN changes over differentiation. The dynamics of the GRN structure can provide us insight into the context-dependent regulatory mechanisms.

The jupyter notebook files and data used in this tutorial are available [here](#).

Python notebook

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
```

```
[2]: import celloracle as co
co.__version__
```

```
[2]: '0.5.0'
```

```
[7]: # visualization settings
%config InlineBackend.figure_format = 'retina'
%matplotlib inline

plt.rcParams['figure.figsize'] = [6, 4.5]
plt.rcParams["savefig.dpi"] = 300
```

0.1. Check installation

Celloracle uses some R libraries in network analysis. Please make sure that all dependent R libraries are installed on your computer. You can test the installation with the following command.

```
[4]: co.network_analysis.test_R_libraries_installation()

R path: /usr/bin/R
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

```
[5]: save_folder = "figures"  
os.makedirs(save_folder, exist_ok=True)
```

1. Load data

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.

```
[6]: # Load data. !!Replace the data path below when you use another data.  
adata = sc.read_h5ad("../03_scRNA-seq_data_preprocessing/data/Paul_et al_15.h5ad")  
  
/home/k/anaconda3/envs/pandas1/lib/python3.6/site-packages/anndata/compat/__init__.py:  
↳161: FutureWarning: Moving element from .uns['neighbors']['distances'] to .obsp[  
↳'distances'].  
  
This is where adjacency matrices should go now.  
FutureWarning,  
/home/k/anaconda3/envs/pandas1/lib/python3.6/site-packages/anndata/compat/__init__.py:  
↳161: FutureWarning: Moving element from .uns['neighbors']['connectivities'] to .  
↳obsp['connectivities'].  
  
This is where adjacency matrices should go now.  
FutureWarning,
```

1.2. Load TF data.

For the GRN inference, celloracle needs TF information, which contains lists of the regulatory candidate genes. There are several ways to make such TF information. We can generate TF information from scATAC-seq data or bulk ATAC-seq data. Please refer to the first step of the tutorial for the details of this process.

If you do not have your scATAC-seq data, you can use some built-in data in celloracle. The built-in TFinfo wqs made using various tissue/cell-types from the mouse ATAC-seq atlas dataset (<http://atlas.gs.washington.edu/mouse-atac/>).

You can load and use the data with the following command.

```
[11]: # 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()  
  
[11]:
```

	peak_id	gene_short_name	9430076c15rik	Ac002126.6	\
0	chr10_100050979_100052296	4930430F08Rik		0.0	0.0
1	chr10_101006922_101007748	SNORA17		0.0	0.0
2	chr10_101144061_101145000	Mgat4c		0.0	0.0
3	chr10_10148873_10149183	9130014G24Rik		0.0	0.0
4	chr10_10149425_10149815	9130014G24Rik		0.0	0.0

	Ac012531.1	Ac226150.2	Afp	Ahr	Ahrr	Aire	...	Znf784	Znf8	Znf816	\
0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

(continues on next page)

(continued from previous page)

2	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
<hr/>										
Znf85	Zscan10	Zscan16	Zscan22	Zscan26	Zscan31	Zscan4				
0	0.0	0.0	0.0	0.0	0.0	0.0				
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0			
2	0.0	0.0	0.0	0.0	0.0	0.0	1.0			
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		

[5 rows x 1095 columns]

2. Initiate Oracle object

Celloracle has a custom called Oracle. We can use Oracle for the data preprocessing and GRN inference steps. The Oracle object stores all of necessary information and does the calculations with its internal functions. We instantiate an Oracle object, then input the gene expression data (anndata) and a TFinfo into the Oracle object.

```
[7]: # Instantiate Oracle object
oracle = co.Oracle()
```

2.1. load gene expression data into oracle object.

When you load a scRNA-seq data, please enter the name of clustering data and dimensional reduction data. The clustering data should be stored in the attribute of “obs” in the anndata. Dimensional reduction data suppose to be stored in the attribute of “obsm” in the anndata. You can check these data by the following command.

If you are not familiar with anndata, please look at the documentation of annata (<https://anndata.readthedocs.io/en/stable/>) or Scanpy (<https://scanpy.readthedocs.io/en/stable/>).

For the celloracle analysis, the anndata shoud include (1) gene expression count, (2) clustering information, (3) trajectory (dimensional reduction embeddings) data. Please refer to another notebook for more information on anndata preprocessing.

```
[8]: # show data name in anndata
print("metadata columns : ", list(adata.obs.columns))
print("dimensional reduction: ", list(adata.obsm.keys()))
metadata columns : ['paul15_clusters', 'n_counts_all', 'n_counts', 'louvain', 'cell_type', 'louvain_annot']
dimensional reduction: ['X_diffmap', 'X_draw_graph_fa', 'X_pca']
```

```
[9]: # In this notebook, we use raw mRNA count as an input of Oracle object.
adata.X = adata.raw.X.copy()

# Instantiate Oracle object.
oracle.import_anndata_as_raw_count(adata=adata,
                                    cluster_column_name="louvain_annot",
                                    embedding_name="X_draw_graph_fa")
```

2.2. Load TFinfo into oracle object

```
[13]: # You can load TF info dataframe with the following code.  
oracle.import_TF_data(TF_info_matrix=TFinfo_df)  
  
# Alternatively, if you saved the information as a dictionary, you can use the code  
# below.  
# oracle.import_TF_data(TFdict=TFinfo_dictionary)
```

2.3. (Optional) Add TF info manually

While we mainly use TF info data made from scATAC-seq data, we can also add additional information about the TF-target gene pair manually.

For example, if there is a study or database that includes specific TF-target pairs, you can use such information in the following way.

2.3.1. Make TF info dictionary manually

Here, we will introduce how to add TF binding information.

We will start with TF binding data from supplemental table 4 in (<http://doi.org/10.1016/j.cell.2015.11.013>).

In order to import TF data into the Oracle object, we need to convert them into a python dictionary. The dictionary keys will be the target genes, and the values will be the regulatory candidate TFs.

```
[50]: # 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
```

	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	Sfpil	0910001L09Rik, 2310014H01Rik, 4632428N05Rik, A...

```
[51]: # Make dictionary: dictionary Key is TF, dictionary Value is list of target genes  
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 invert the dictionary above using a utility function in celloracle.  
TG_to_TF_dictionary = co.utility.inverse_dictionary(TF_to_TG_dictionary)  
  
HBox(children=(FloatProgress(value=0.0, max=178.0), HTML(value='')))
```

2.3.2. Add TF information dictionary into the oracle object

```
[53]: # Add TF information
oracle.addTFinfo_dictionary(TG_to_TF_dictionary)
```

3. Knn imputation

Celloracle uses almost the same strategy as velocyto for visualizing cell transitions. This process requires KNN imputation in advance.

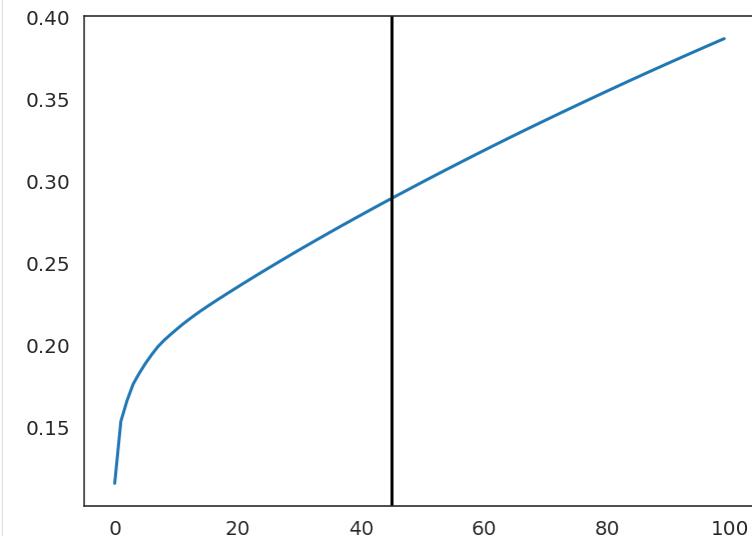
For the KNN imputation, we need PCA and PC selection first.

3.1. PCA

```
[60]: # 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.
                           ↪ 0.02)) [0] [0]
plt.axvline(n_comps, c="k")
print(n_comps)
n_comps = min(n_comps, 50)
```

45



3.2. KNN imputation

Estimate the optimal number of nearest neighbors for KNN imputation.

```
[63]: n_cell = oracle.adata.shape[0]
print(f"cell number is :{n_cell}")
cell number is :2671
```

```
[64]: k = int(0.025*n_cell)
print(f"Auto-selected k is :{k}")
Auto-selected k is :66
```

```
[65]: 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.

Celloracle has some custom-classes: Links, Oracle and TFinfo. You can save such an object using “to_hdf5”.

Please use “load_hdf5” function to load the file.

```
[66]: # Save oracle object.
oracle.to_hdf5("Paul_15_data.celloracle.oracle")
```

```
[19]: # Load file.
#oracle = co.load_hdf5("Paul_15_data.celloracle.oracle")
```

5. GRN calculation

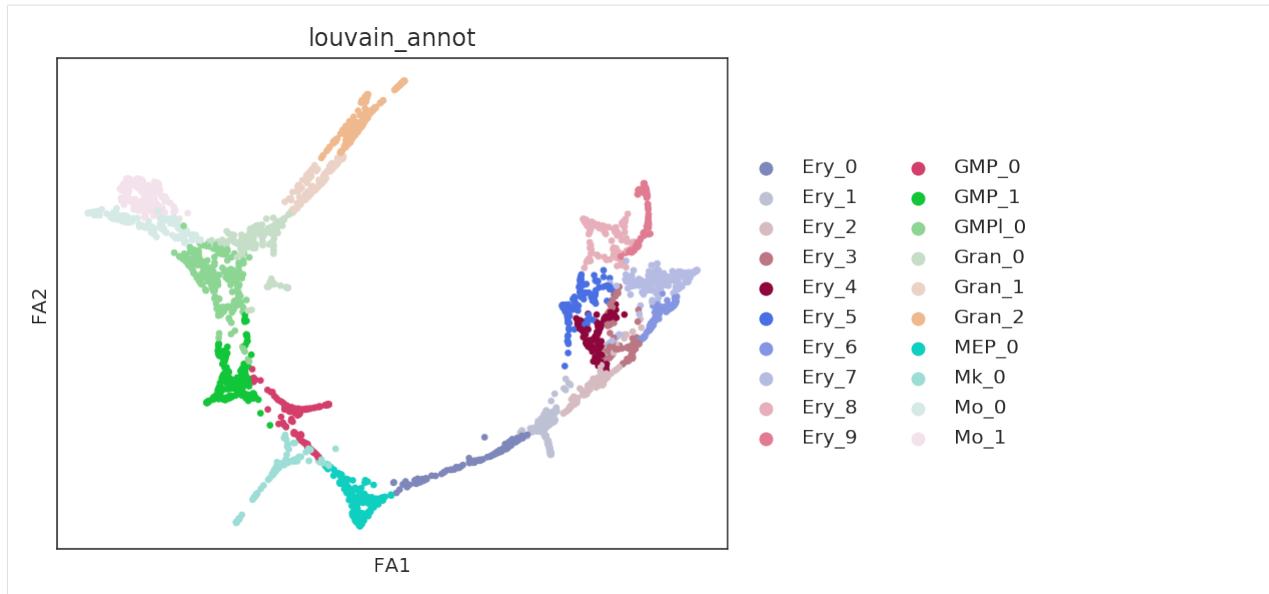
The next step is constructing a cluster-specific GRN for all clusters.

You can calculate GRNs with the “get_links” function, and the function returns GRNs as a Links object. The Links object stores inferred GRNs and the corresponding metadata. You can do network analysis with the Links object.

The GRN will be calculated for each cluster/sub-group. In the example below, we construct GRN for each unit of the “louvain_annot” clustering.

The GRNs can be calculated at any arbitrary unit as long as the clustering information is stored in anndata.

```
[67]: # check data
sc.pl.draw_graph(oracle.adata, color="louvain_annot")
```



5.1. Get GRNs

```
[ ]: %%time
# 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=10, test_mode=False)
```

5.2. (Optional) Export GRNs

Although celloracle has many functions for network analysis, you can analyze GRNs by hand if you choose. The raw GRN data is stored in the attribute of “links_dict”.

For example, you can get the GRN for the “Ery_0” cluster with the following commands.

```
[72]: links.links_dict.keys()
[72]: dict_keys(['Ery_0', 'Ery_1', 'Ery_2', 'Ery_3', 'Ery_4', 'Ery_5', 'Ery_6', 'Ery_7',
   ↪'Ery_8', 'Ery_9', 'GMP_0', 'GMP_1', 'GMPI_0', 'Gran_0', 'Gran_1', 'Gran_2', 'MEP_0',
   ↪ 'Mk_0', 'Mo_0', 'Mo_1'])
```

```
[73]: links.links_dict["Ery_0"]
[73]:      source      target  coef_mean  coef_abs      p      -logp
0       Stat3  0610007L01Rik  -0.010275  0.010275  3.476931e-07  6.458804
1       Gata1  0610007L01Rik  -0.000380  0.000380  7.598357e-01  0.119280
2       Zbtb1  0610007L01Rik   0.004452  0.004452  1.018526e-03  2.992028
3       Rara  0610007L01Rik  -0.000669  0.000669  7.065405e-01  0.150863
4        Myc  0610007L01Rik  -0.010705  0.010705  1.696471e-05  4.770454
...
...      ...      ...      ...      ...      ...      ...      ...
(continues on next page)
```

(continued from previous page)

74420	Smarcc2	Zyx	-0.003475	0.003475	2.754236e-02	1.559999
74421	Nfe2	Zyx	0.031430	0.031430	1.461503e-11	10.835200
74422	Zbtb4	Zyx	0.001684	0.001684	1.915555e-01	0.717705
74423	Smarcc1	Zyx	0.011356	0.011356	1.843519e-04	3.734352
74424	Nfkbl	Zyx	0.010803	0.010803	1.805959e-06	5.743292

[74425 rows x 6 columns]

You can export the file as follows.

```
[ ]: # Set cluster name
cluster = "Ery_0"

# Save as csv
links.links_dict[cluster].to_csv(f"raw_GRN_for_{cluster}.csv")
```

5.3. (Optional) Change order

The links object has a color information in an attribute, “palette”. This information is used for the visualization

The sample will be visualized in that order. Here we can change the order.

```
[75]: # Show the contents of palette
links.palette
```

```
[75]: 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
GMP1_0     #8DD593
Mo_0       #D5EAE7
Mo_1       #F3E1EB
Gran_0     #C6DEC7
Gran_1     #EAD3C6
Gran_2     #F0B98D
```

```
[76]: # Change the order of palette
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',
         'GMP1_0', 'Mo_0', 'Mo_1', 'Gran_0', 'Gran_1', 'Gran_2']
links.palette = links.palette.loc[order]
links.palette
```

```
[76]: palette
MEP_0      #0FCFC0
```

(continues on next page)

(continued from previous page)

```
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
GMP1_0   #8DD593
Mo_0     #D5EAE7
Mo_1     #F3E1EB
Gran_0   #C6DEC7
Gran_1   #EAD3C6
Gran_2   #F0B98D
```

6. Network preprocessing

6.1. Filter network edges

Celloracle utilizes bagging ridge or Bayesian ridge regression to infer gene regulatory networks. These methods provide a network edge strength as a distribution rather than a point value. We can use the distribution to know the certainness of the connection.

We filter the network edges as follows.

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

The raw network data is stored as an attribute, “links_dict,” while filtered network data is stored in “filtered_links.” Thus the filtering function keeps raw network information rather than overwriting the data. You can come back to the filtering process to filter the data with different parameters if you want.

```
[78]: links.filter_links(p=0.001, weight="coef_abs", thread_number=2000)
```

6.2. Degree distribution

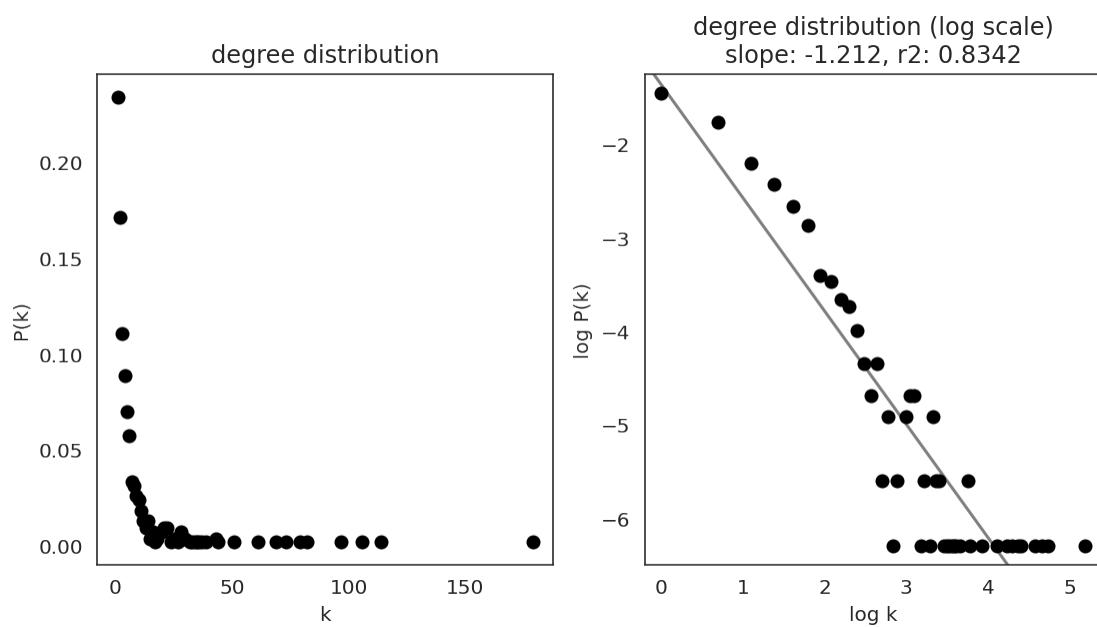
In the first step, we examine the network degree distribution. Network degree, which is the number of edges for each node, is one of the important metrics used to investigate the network structure (https://en.wikipedia.org/wiki/Degree_distribution).

Please keep in mind that the degree distribution may change depending on the filtering threshold.

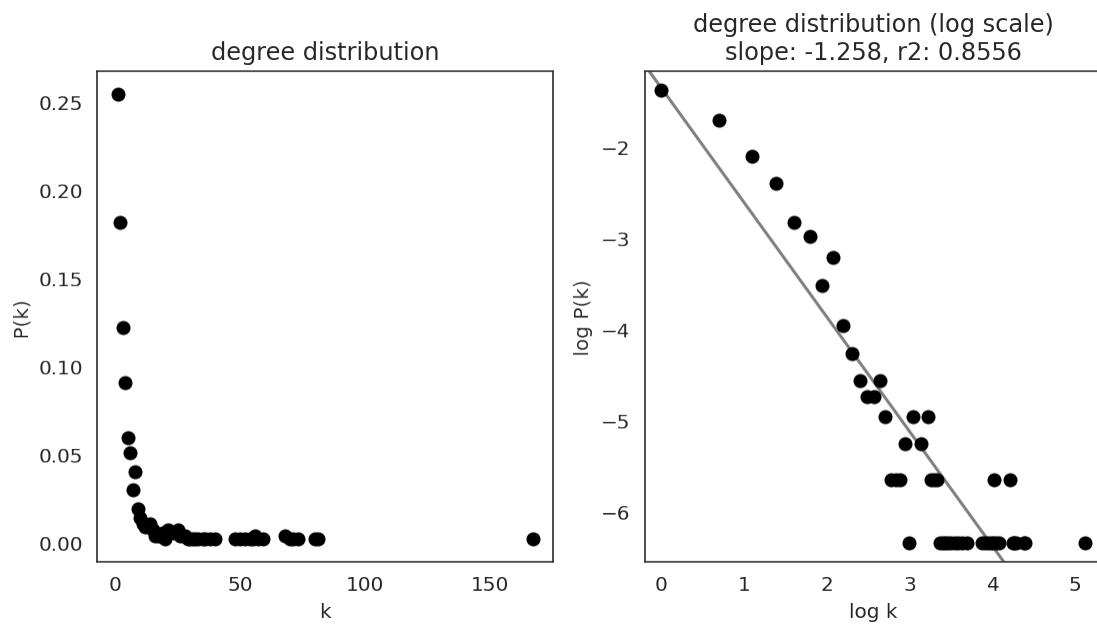
```
[9]: plt.rcParams["figure.figsize"] = [9, 4.5]
```

```
[10]: links.plot_degree_distributions(plot_model=True,
                                     #save=f"{save_folder}/degree_
                                     ↪distribution/",
                                     )
```

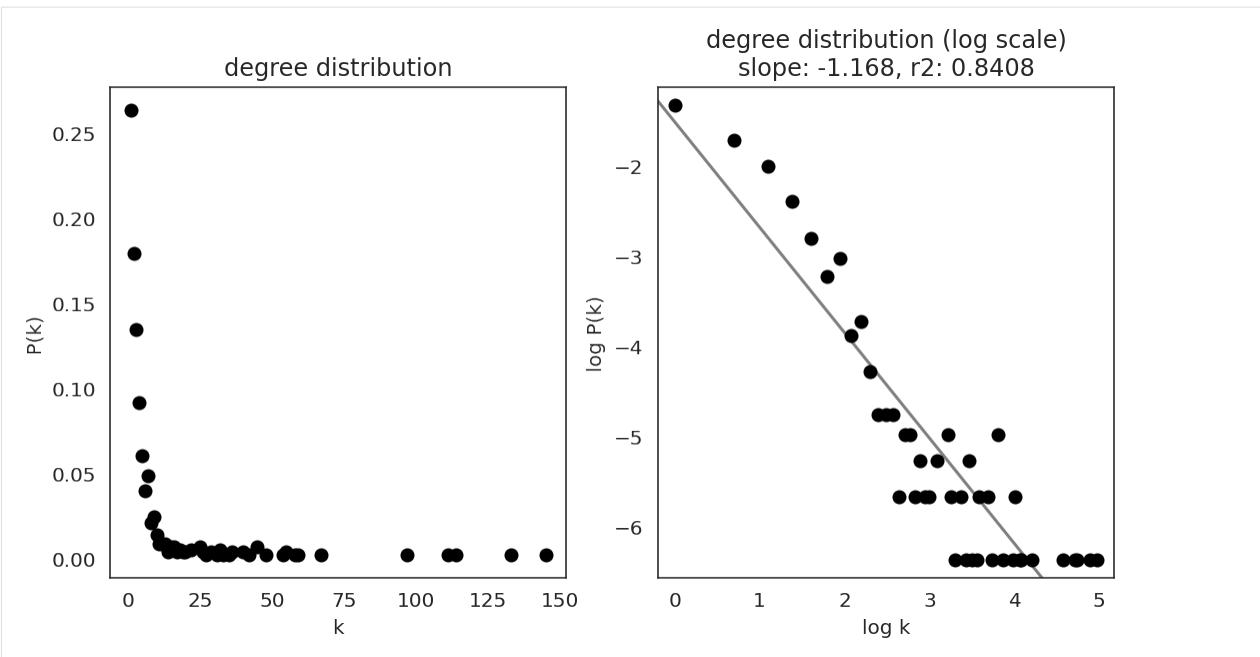
Ery_0



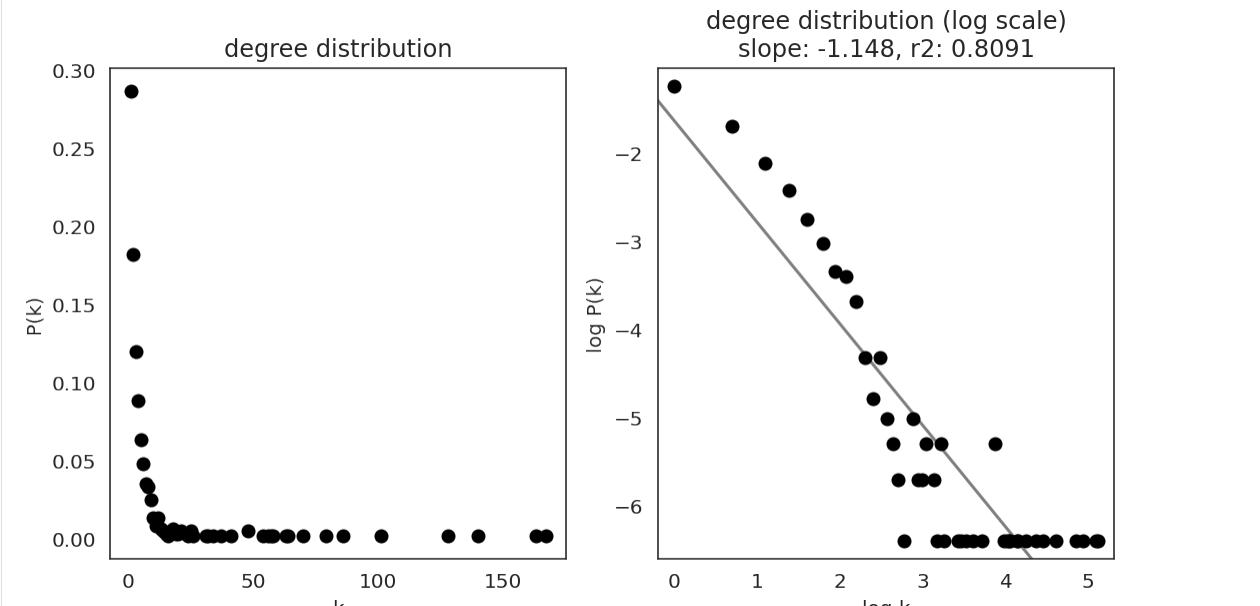
Ery_1



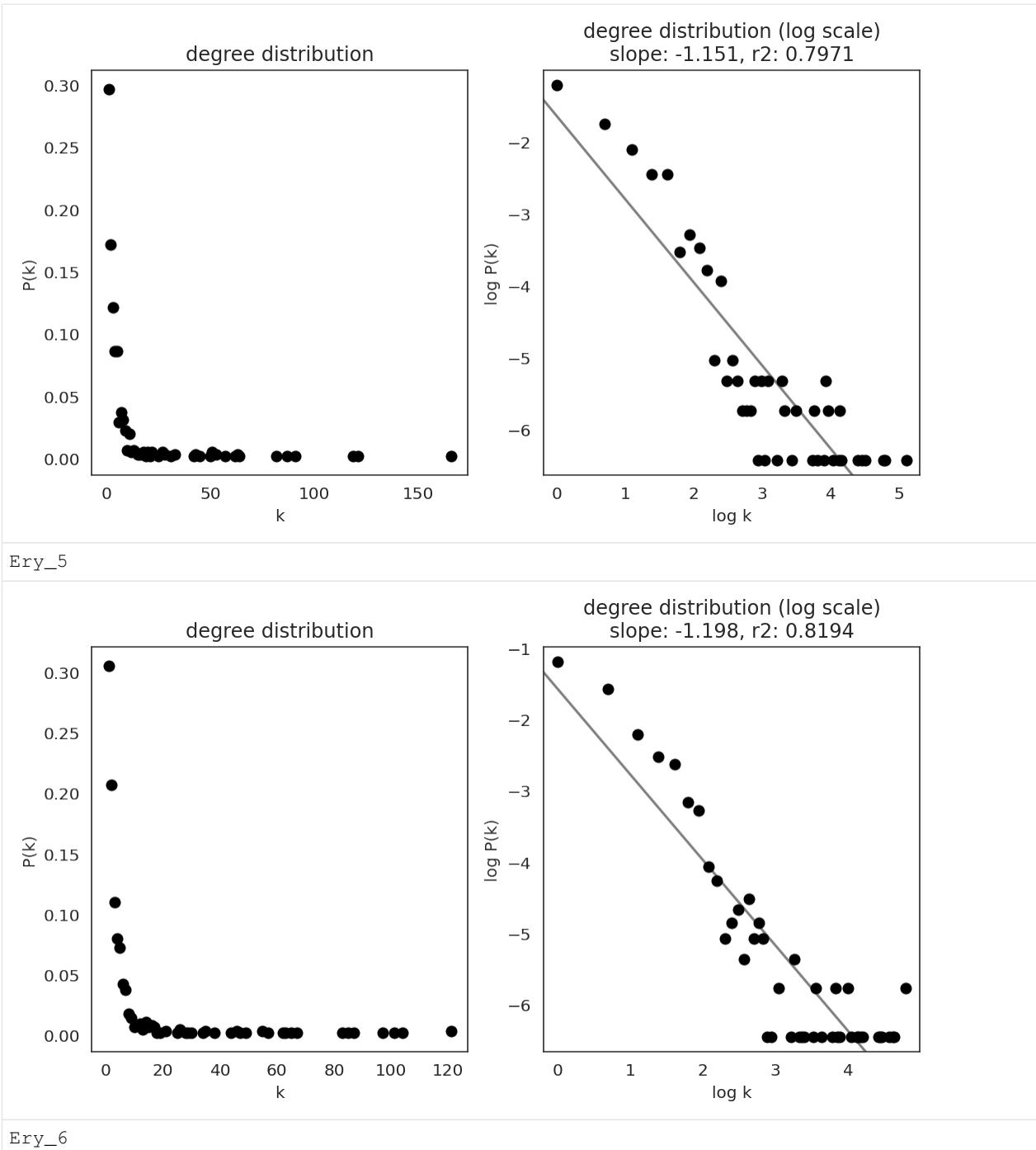
Ery_2

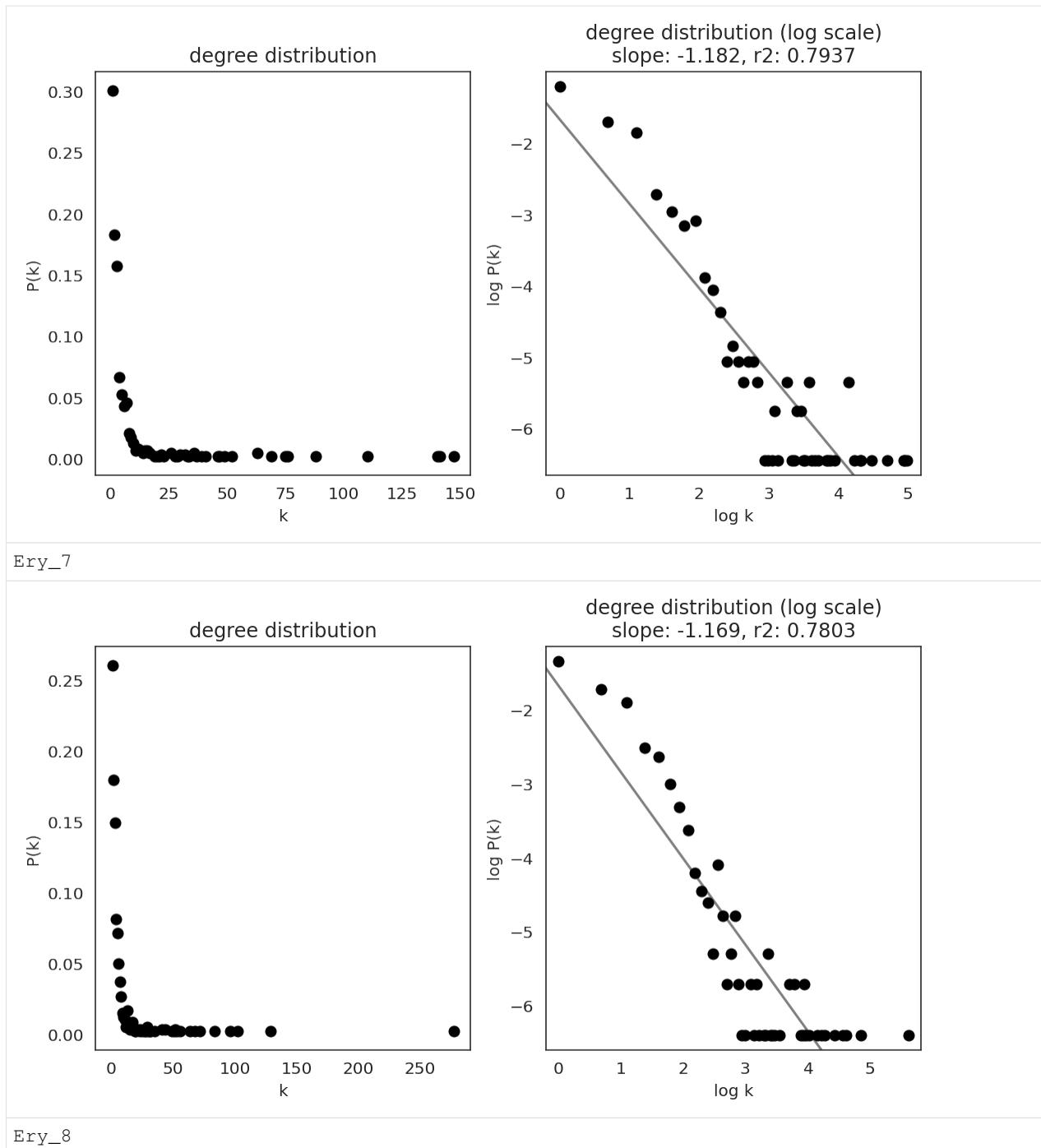


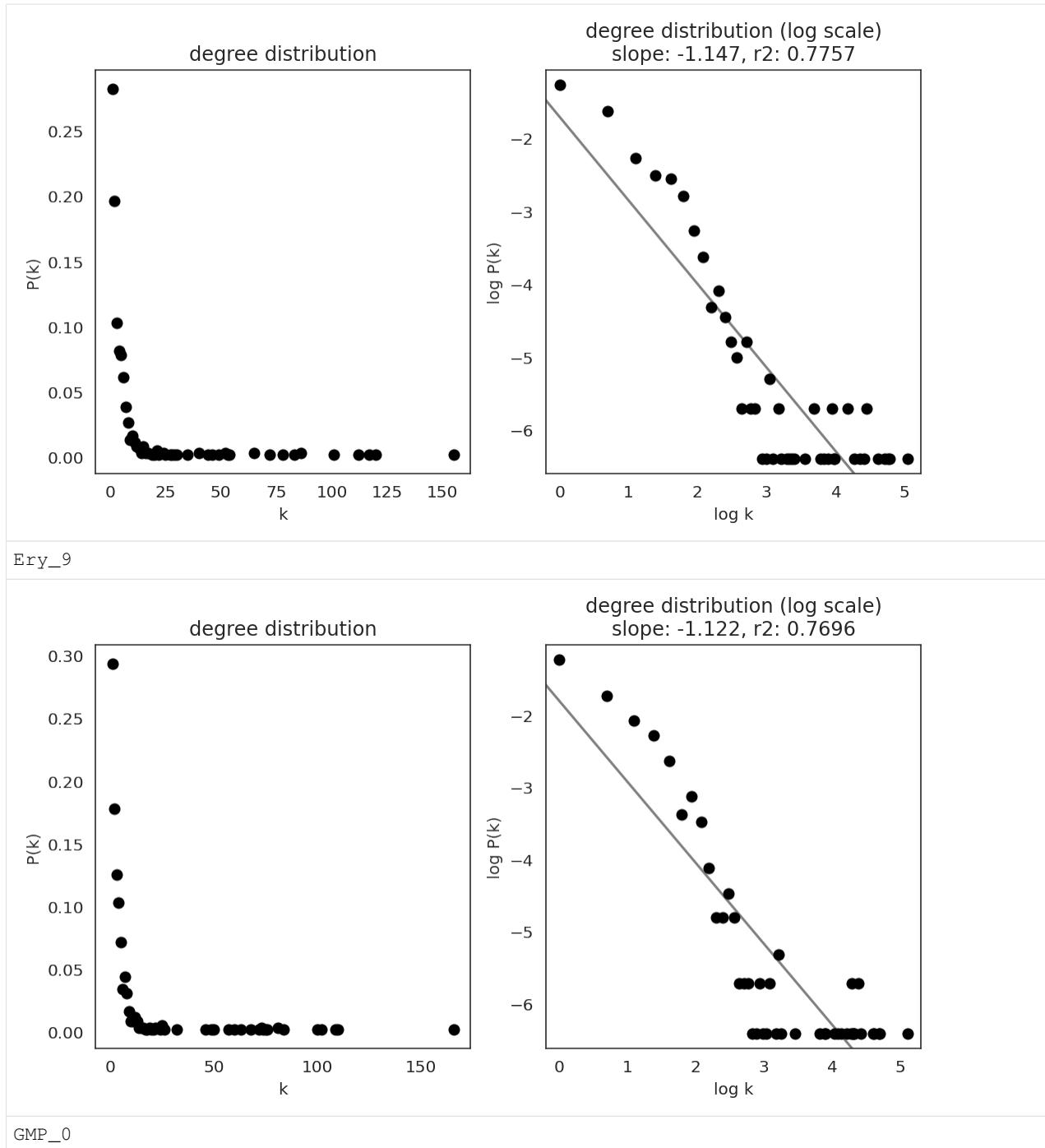
Ery_3

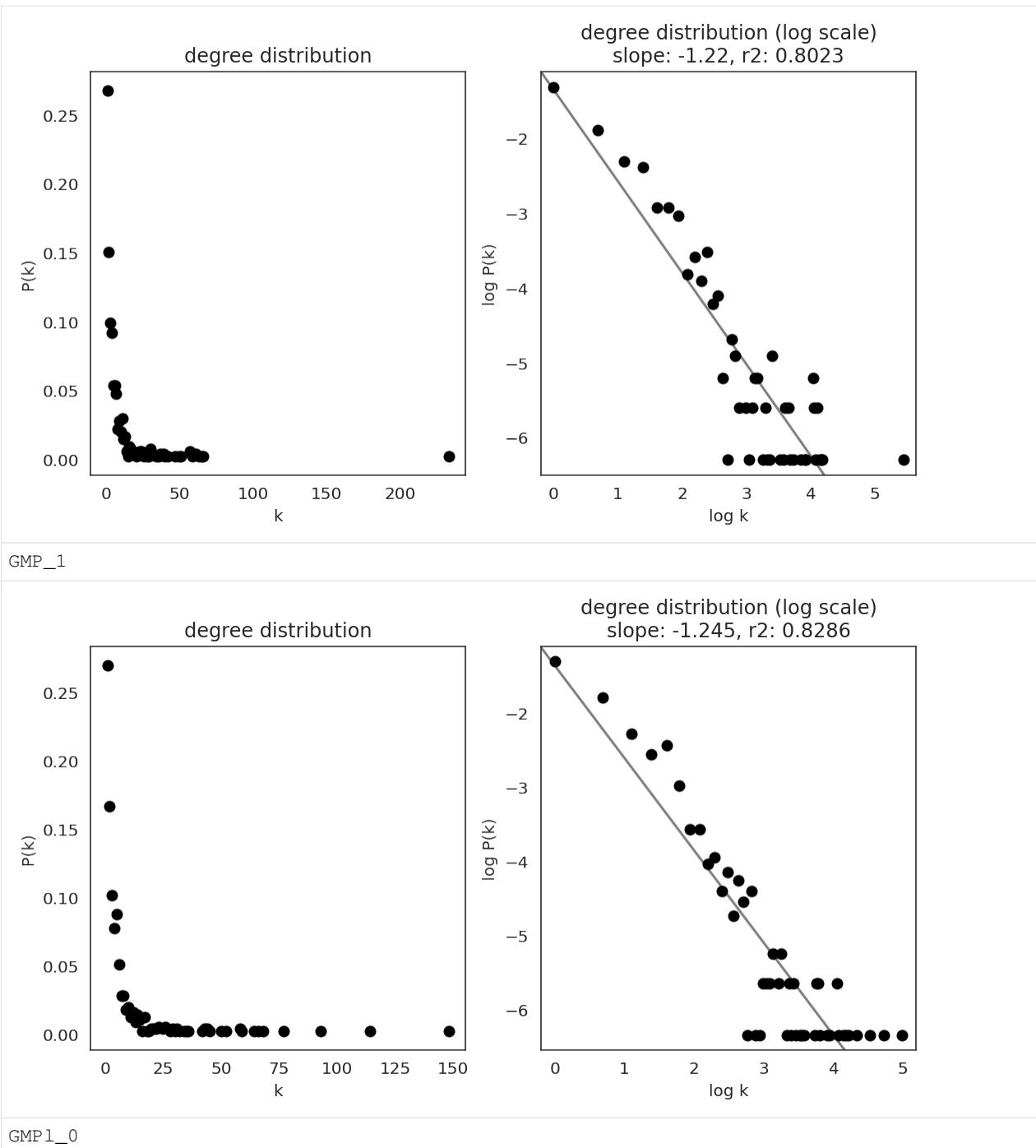


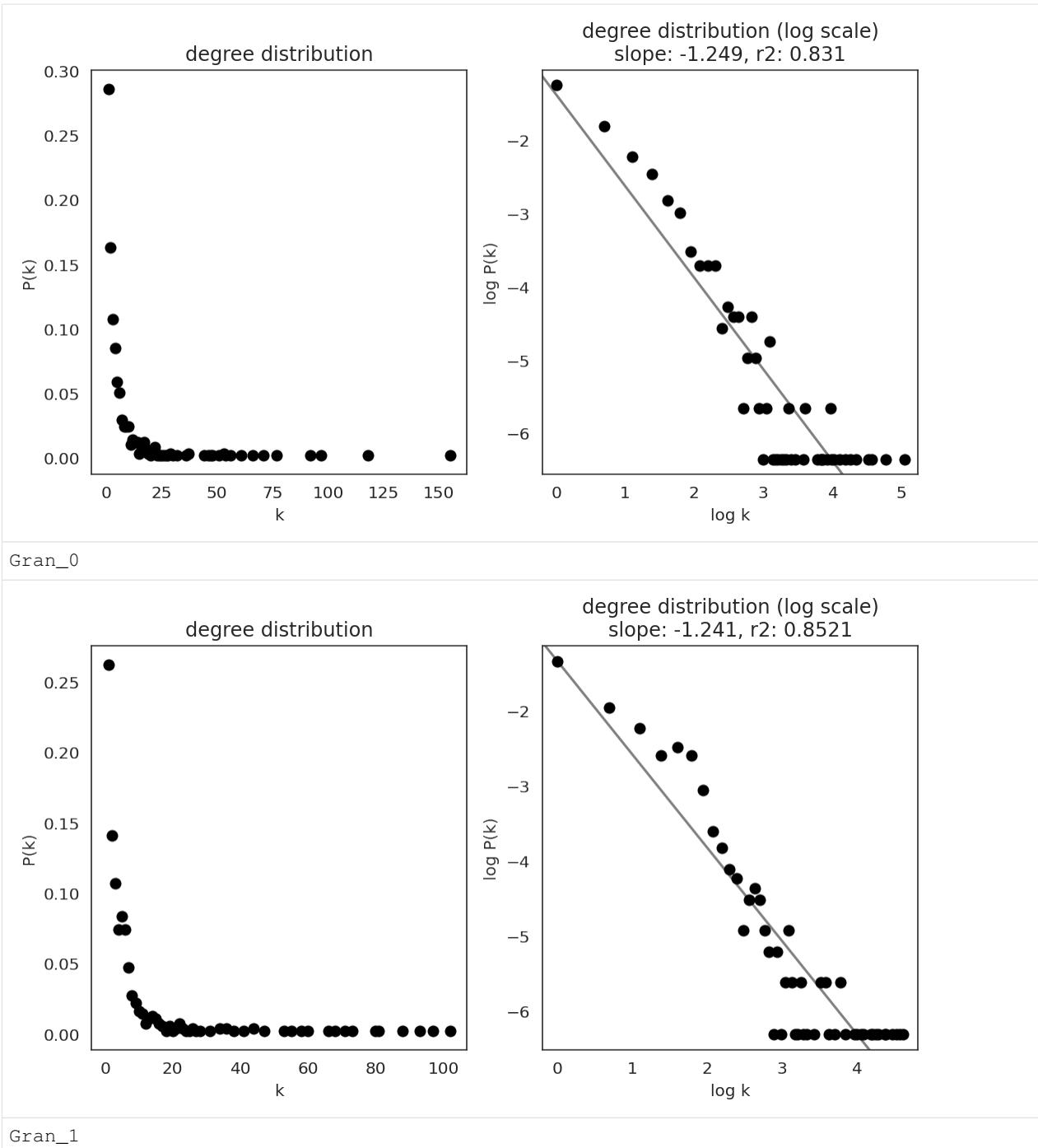
Ery_4

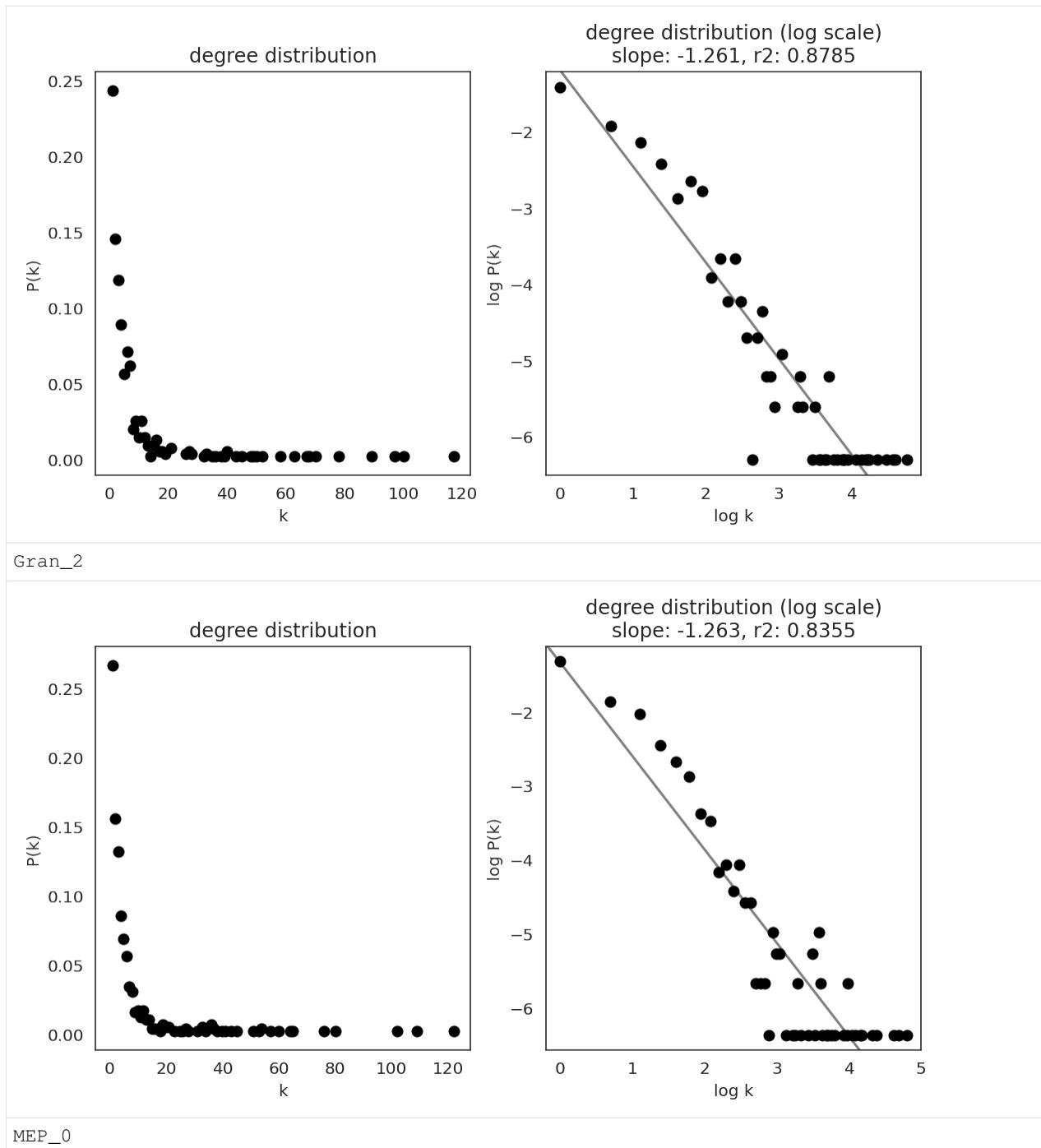


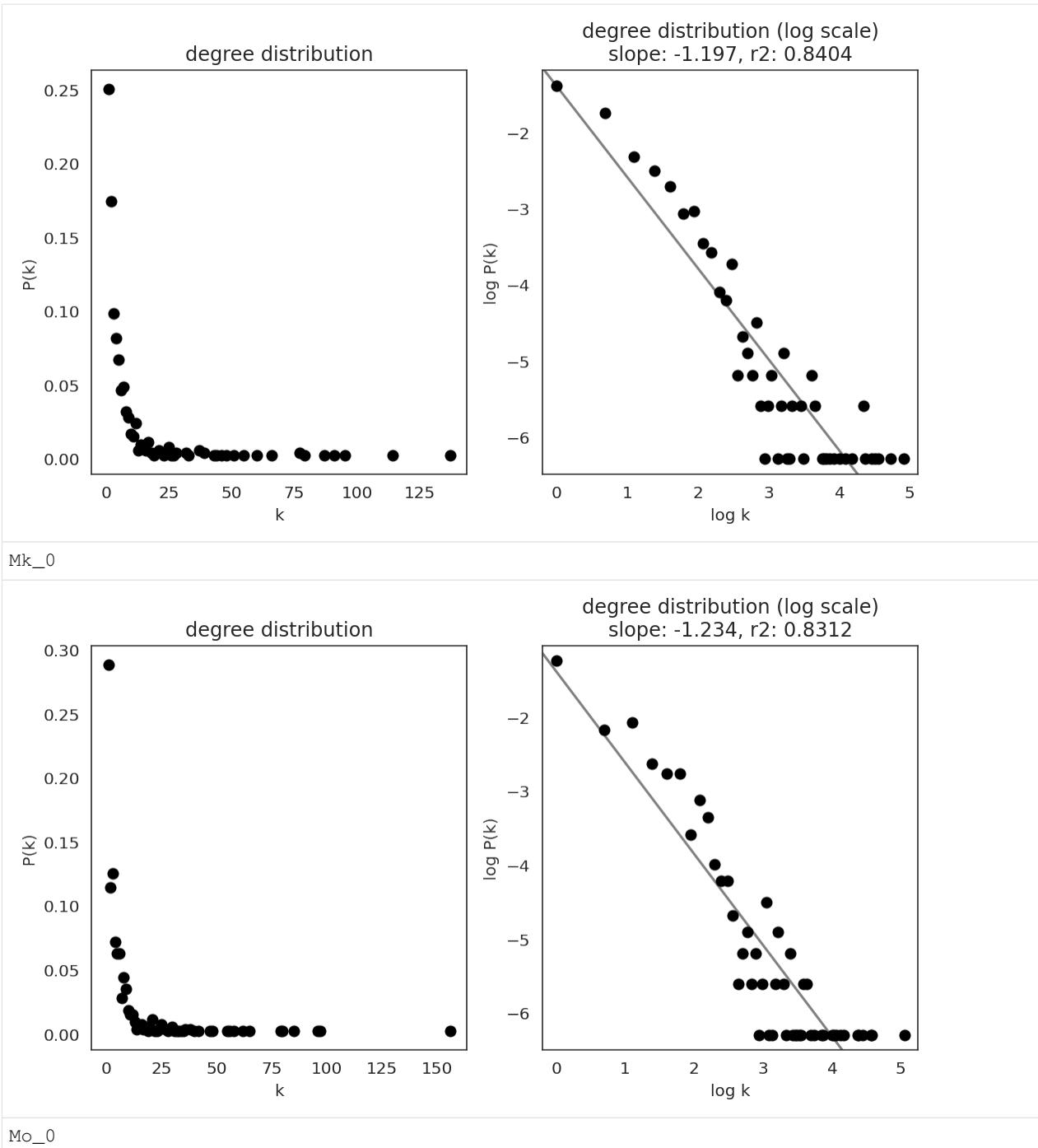


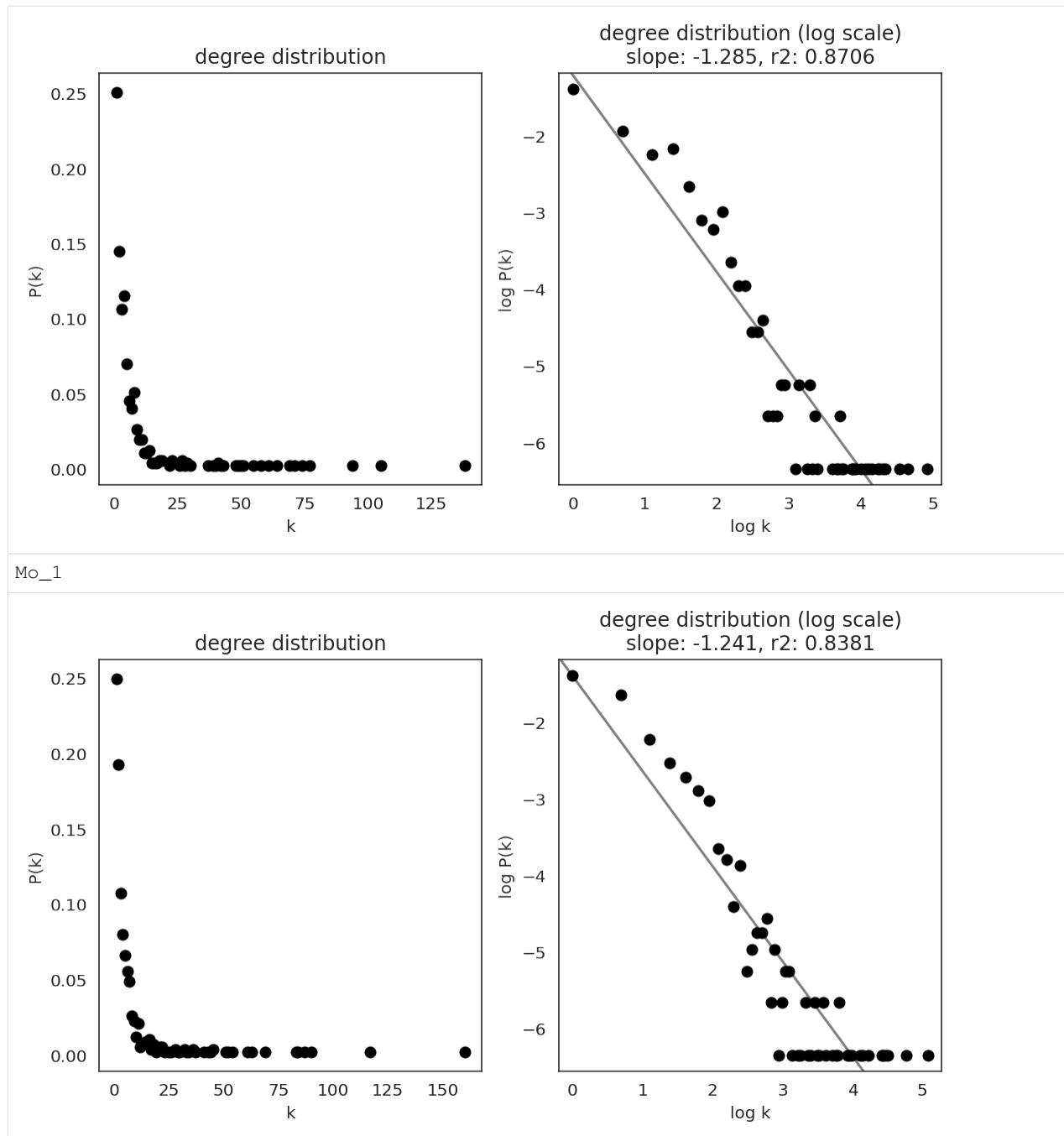












```
[28]: plt.rcParams["figure.figsize"] = [6, 4.5]
```

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.

```
[25]: # Calculate network scores. It takes several minutes.
links.get_score()

processing... batch 1/3
Ery_0: finished.
Ery_1: finished.
Ery_2: finished.
Ery_3: finished.
Ery_4: finished.
Ery_5: finished.
Ery_6: finished.
Ery_7: finished.
processing... batch 2/3
Ery_8: finished.
Ery_9: finished.
GMP_0: finished.
GMP_1: finished.
GMP1_0: finished.
Gran_0: finished.
Gran_1: finished.
Gran_2: finished.
processing... batch 3/3
MEP_0: finished.
Mk_0: finished.
Mo_0: finished.
Mo_1: finished.
```

The score is stored as a attribute called “merged_score”, and the score will also be saved in a folder in your computer.

```
[82]: links.merged_score.head()

[82]:   degree_all  degree_in  degree_out  clustering_coefficient \
Stat3        82         0         82          0.021981
Mycn        30         0         30          0.011494
E2f4       181         2        179          0.009822
Zbtb1        22         0         22          0.000000
Ybx1        69         9         60          0.028133

  clustering_coefficient_weighted  degree_centrality_all \
Stat3                  0.022055          0.151292
Mycn                  0.009986          0.055351
E2f4                  0.011874          0.333948
Zbtb1                  0.000000          0.040590
Ybx1                  0.027709          0.127306

  degree_centrality_in  degree_centrality_out  betweenness_centrality \
Stat3          0.000000          0.151292              0
Mycn          0.000000          0.055351              0
E2f4          0.003690          0.330258            3158
Zbtb1          0.000000          0.040590              0
Ybx1          0.016605          0.110701            1051

  closeness_centrality  eigenvector_centrality  page_rank \

```

(continues on next page)

(continued from previous page)

Stat3	0.000012	0.487978	0.001633	
Mycn	0.000010	0.245650	0.001633	
E2f4	0.000010	1.000000	0.001724	
Zbtb1	0.000004	0.113727	0.001633	
Ybx1	0.000004	0.385376	0.002133	
 assortative_coefficient average_path_length community_random_walk \\\				
Stat3	-0.161693	2.621324	1	
Mycn	-0.161693	2.621324	1	
E2f4	-0.161693	2.621324	1	
Zbtb1	-0.161693	2.621324	18	
Ybx1	-0.161693	2.621324	2	
 module connectivity participation role cluster				
Stat3	0	3.573858	Connector Hub	Ery_0
Mycn	2	1.767680	Peripheral	Ery_0
E2f4	0	8.575037	Connector Hub	Ery_0
Zbtb1	2	1.317952	Peripheral	Ery_0
Ybx1	3	5.306800	Connector Hub	Ery_0

6.4. Save

Save processed GRN. We use this file in the next notebook; “in silico perturbation with GRNs”.

```
[31]: # Save Links object.
links.to_hdf5(file_path="links.celloracle.links")
```

```
[6]: # You can load files with the following command.
links = co.load_hdf5(file_path="links.celloracle.links")
```

7. Network analysis; Network score for each gene

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

7.1. Network score in each cluster

We have calculated several network scores using different centrality metrics. We can use the centrality score to identify key regulatory genes because centrality is one of the important indicators of network structure (<https://en.wikipedia.org/wiki/Centrality>).

Let’s visualize genes with high network centrality.

```
[83]: # Check cluster name
links.cluster
```

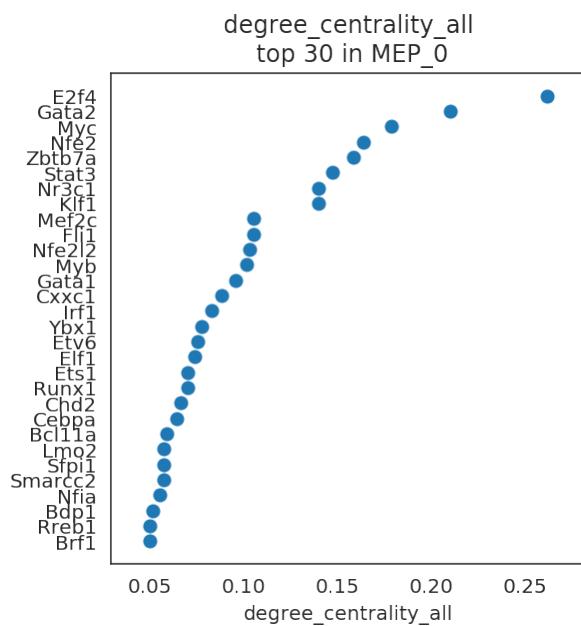
```
[83]: ['Ery_0',
'Ery_1',
'Ery_2',
'Ery_3',
'Ery_4',
'Ery_5',
```

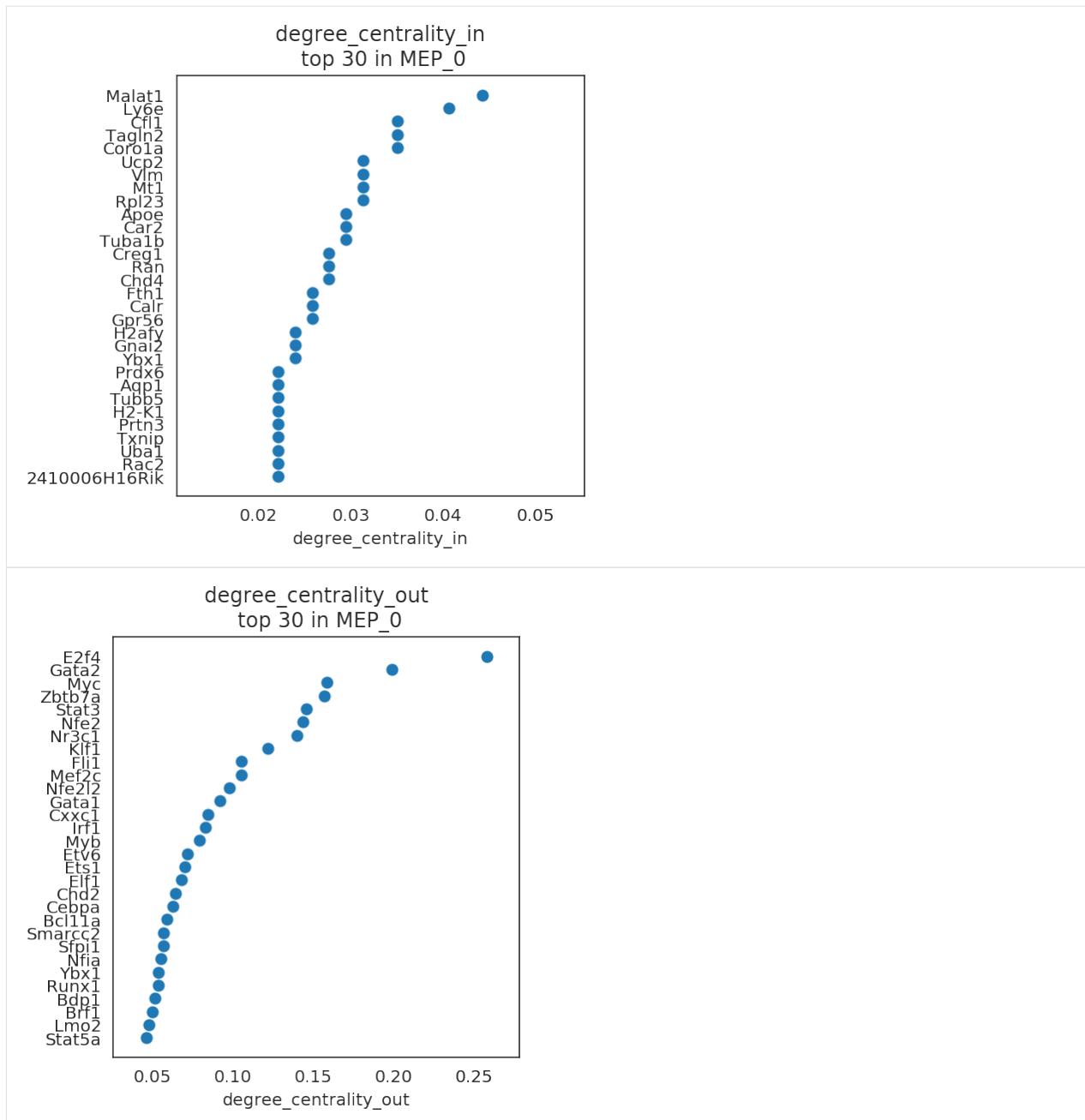
(continues on next page)

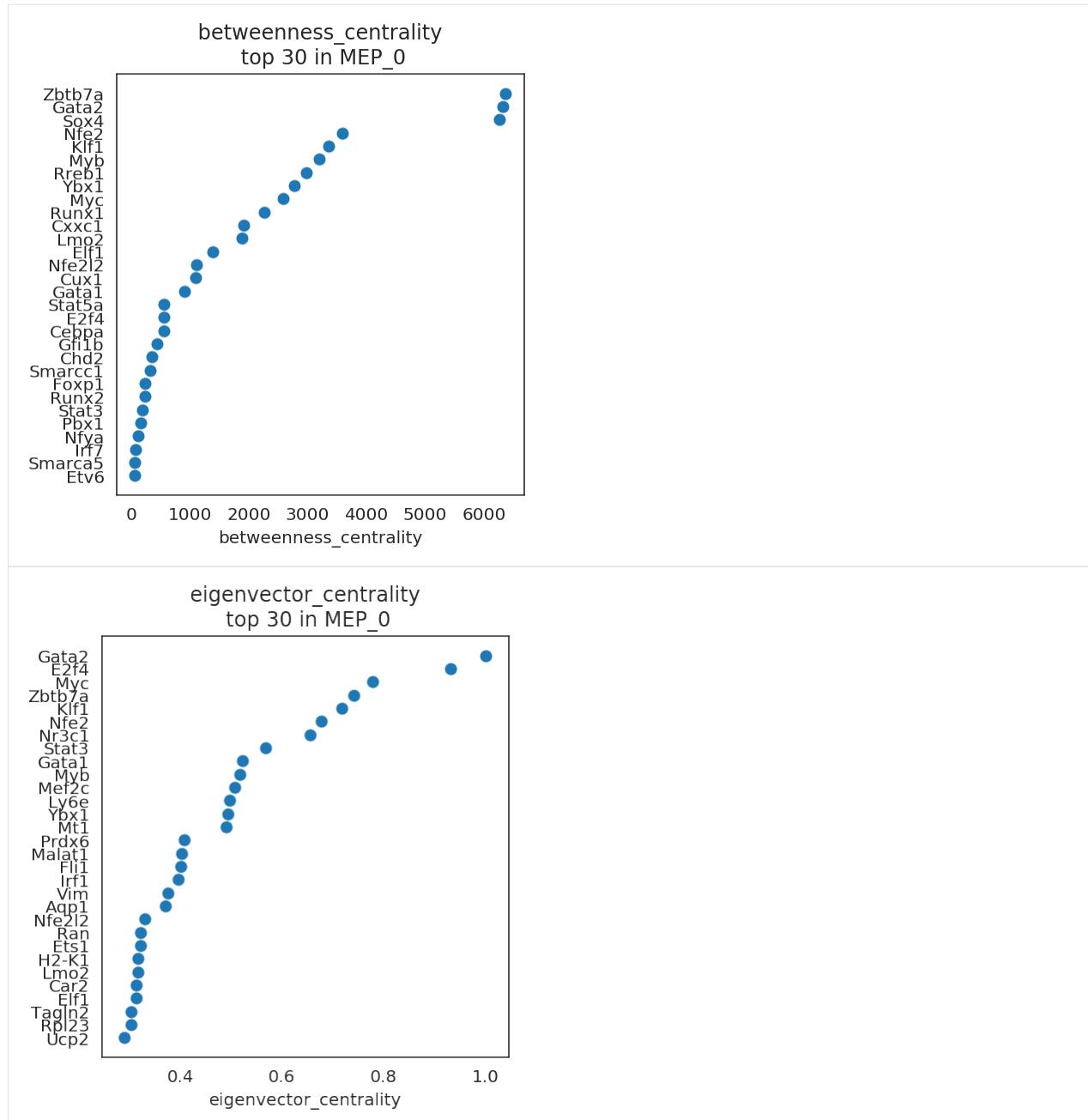
(continued from previous page)

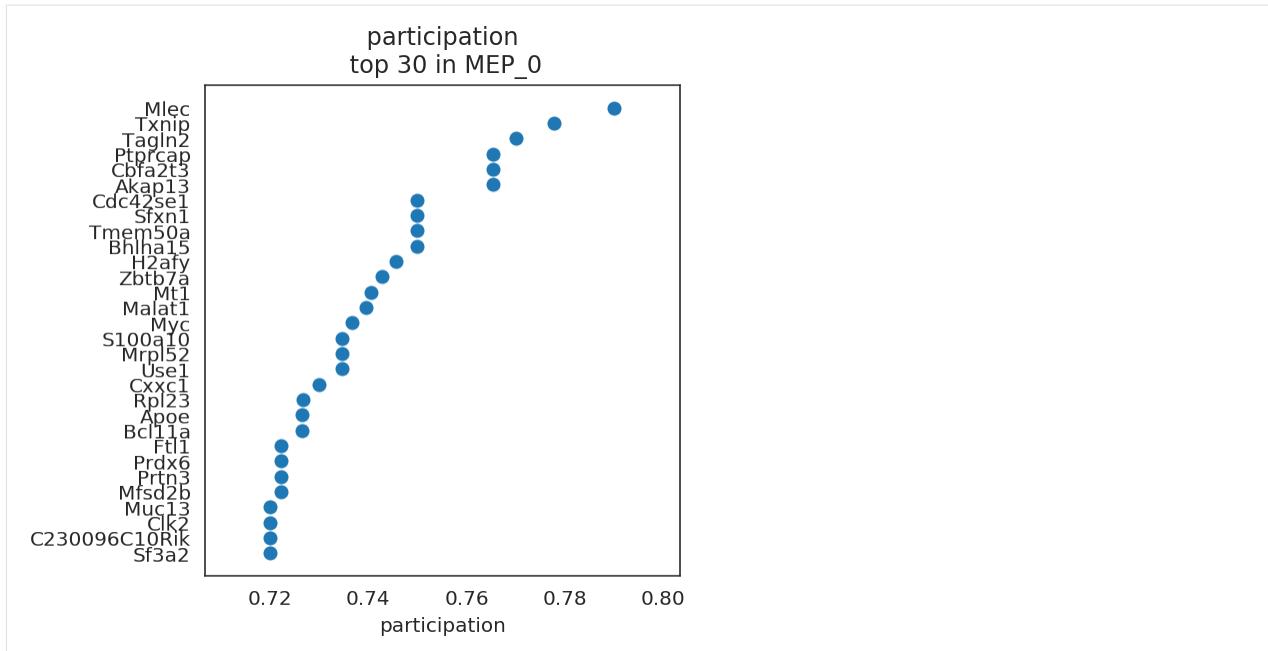
```
'Ery_6',
'Ery_7',
'Ery_8',
'Ery_9',
'GMP_0',
'GMP_1',
'GMP1_0',
'Gran_0',
'Gran_1',
'Gran_2',
'MEP_0',
'Mk_0',
'Mo_0',
'Mo_1']
```

[53]: # 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")





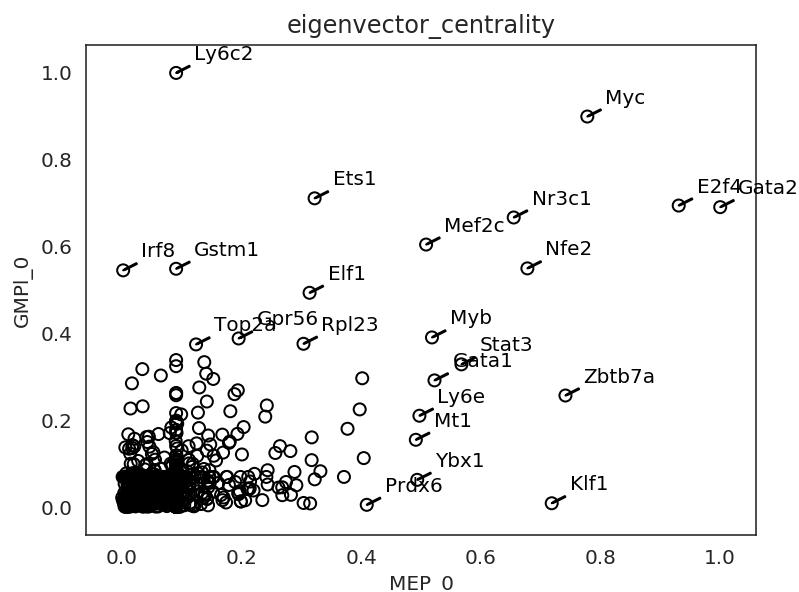




7.2. Network score comparison between two clusters

By comparing network scores between two clusters, we can analyze differences in GRN structure.

```
[54]: plt.ticklabel_format(style='sci',axis='y',scilimits=(0,0))
links.plot_score_comparison_2D(value="eigenvector_centrality",
                                cluster1="MEP_0", cluster2="GMP1_0",
                                percentile=98, save=f'{save_folder}/score_comparison')
```

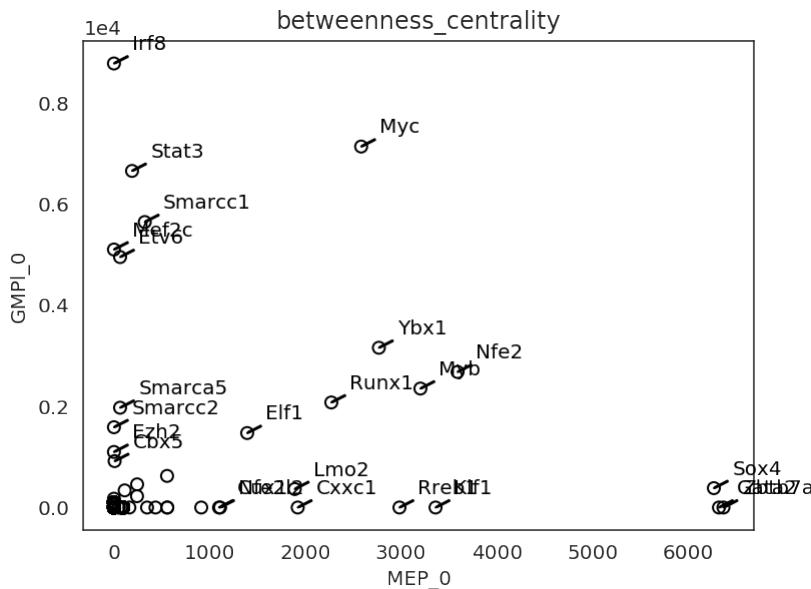


```
[55]: plt.ticklabel_format(style='sci',axis='y',scilimits=(0,0))
```

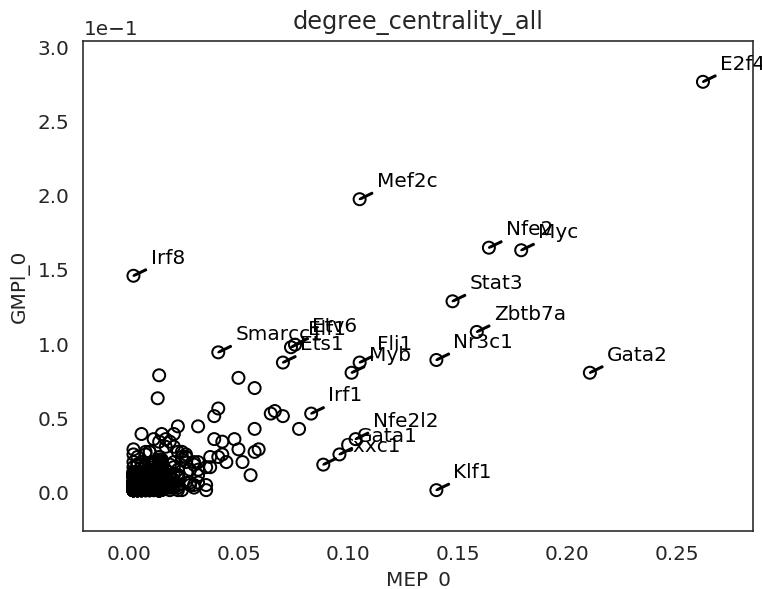
(continues on next page)

(continued from previous page)

```
links.plot_score_comparison_2D(value="betweenness_centrality",
                                cluster1="MEP_0", cluster2="GMP1_0",
                                percentile=98, save=f"{save_folder}/score_comparison")
```



```
[56]: plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
links.plot_score_comparison_2D(value="degree_centrality_all",
                                cluster1="MEP_0", cluster2="GMP1_0",
                                percentile=98, save=f"{save_folder}/score_comparison")
```

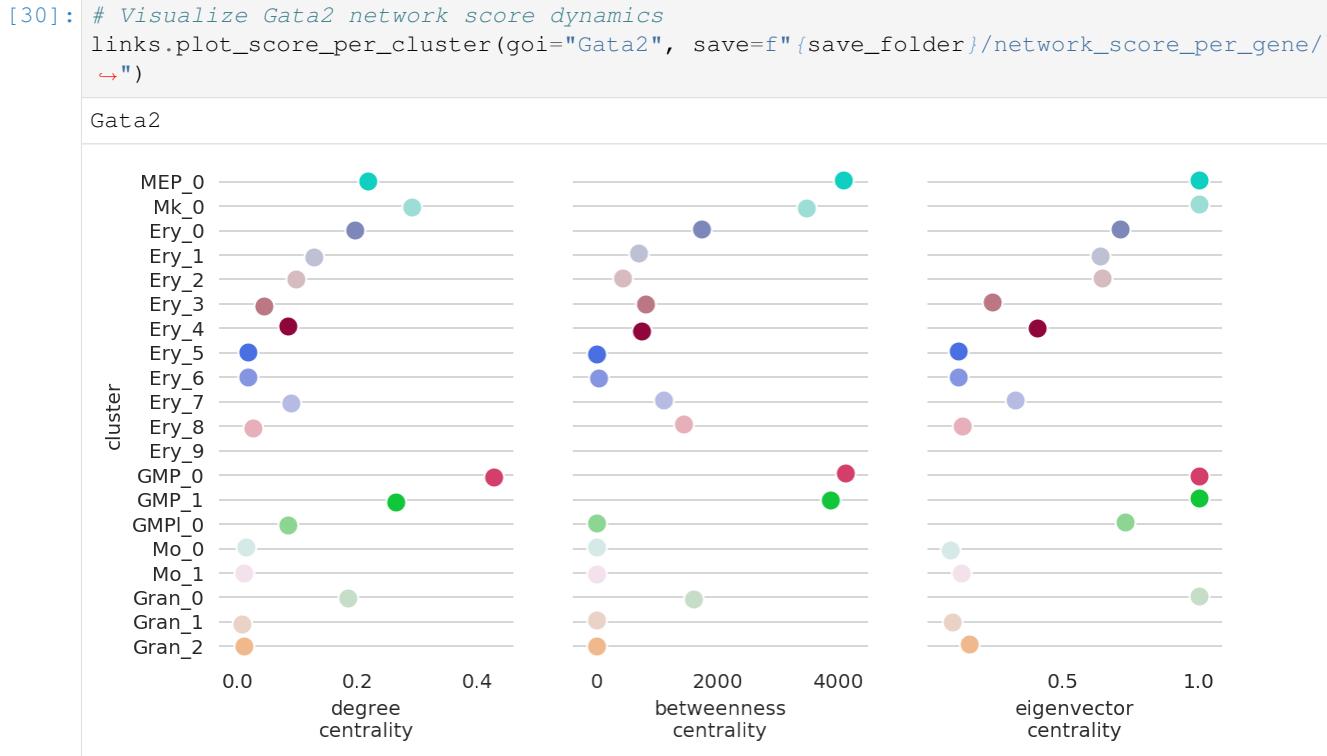


7.3. Network score dynamics

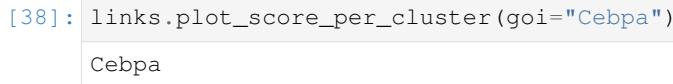
In the following session, we focus on how a gene's network score changes during the differentiation.

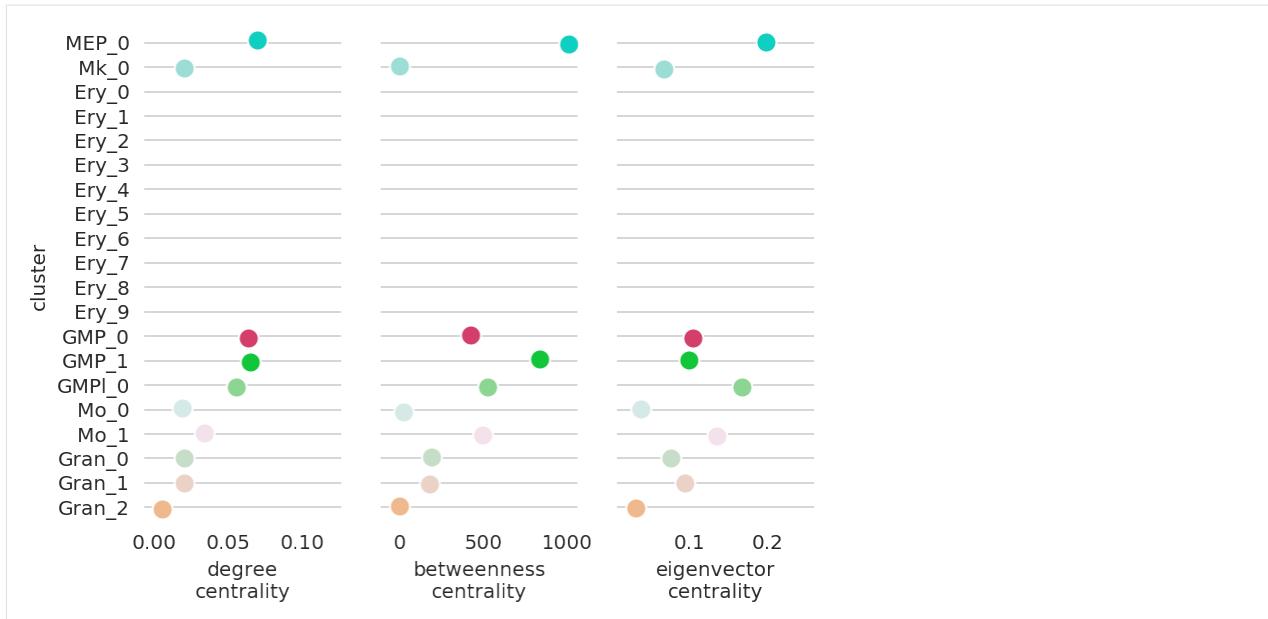
Using Gata2, we will demonstrate how you can visualize networks scores for a single gene.

Gata2 is known to play an essential role in the early MEP and GMP populations. .



If a gene have no connections in a cluster, it is impossible to calculate network degree scores. Thus the scores will not be shown. For example, Cebpa have no connection in the erythroids clusters, and there is no degree scores for Cebpa in these clusters as follows.





You can check filtered network edge as follows.

```
[39]: cluster_name = "Ery_0"
filtered_links_df = links.filtered_links[cluster_name]
filtered_links_df.head()
```

	source	target	coef_mean	coef_abs	p	-logp
68775	Stat3	Top2a	-0.107635	0.107635	1.976987e-14	13.703996
51655	Mycn	Prdx6	-0.096651	0.096651	8.076169e-11	10.092795
41345	Mycn	Mt1	-0.093897	0.093897	8.228218e-15	14.084694
5136	Ybx1	Anp32b	0.089403	0.089403	4.498303e-14	13.346951
41326	E2f4	Mt1	0.089261	0.089261	7.447929e-10	9.127964

You can confirm that there is no Cebpa connection in Ery_0 cluster.

```
[41]: filtered_links_df[filtered_links_df.source == "Cebpa"]
```

	source	target	coef_mean	coef_abs	p	-logp
	Empty DataFrame					
	Columns:	[source, target, coef_mean, coef_abs, p, -logp]				
	Index:	[]				

7.4. Gene cartography analysis

Gene cartography is a method for gene network analysis. The method classifies gene into several groups using the network module structure and connections. It provides us an insight about the role and regulatory mechanism for each gene. For more information on gene cartography, please refer to the following paper (<https://www.nature.com/articles/nature03288>).

The gene cartography will be calculated for the GRN in each cluster. Thus we can know how the gene cartography change by comparing the the score between clusters.

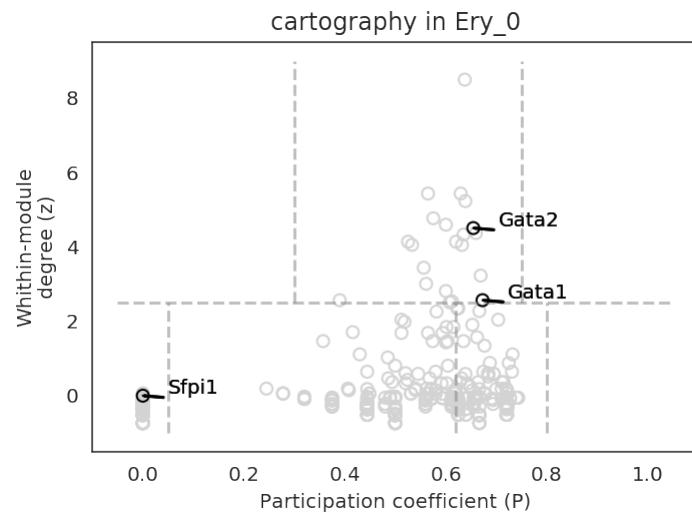
```
[58]: # Plot cartography as a scatter plot
links.plot_cartography_scatter_per_cluster(scatter=True,
                                            kde=False,
                                            gois=["Gata1", "Gata2", "Sfpil1"],
```

(continues on next page)

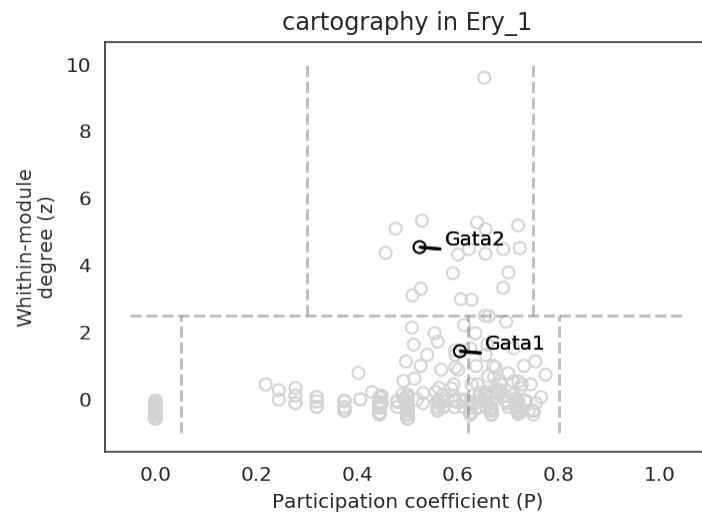
(continued from previous page)

```
auto_gene_annot=False,
args_dot={"n_levels": 105},
args_line={"c":"gray"}, save=f" {save_
→folder}/cartography")
```

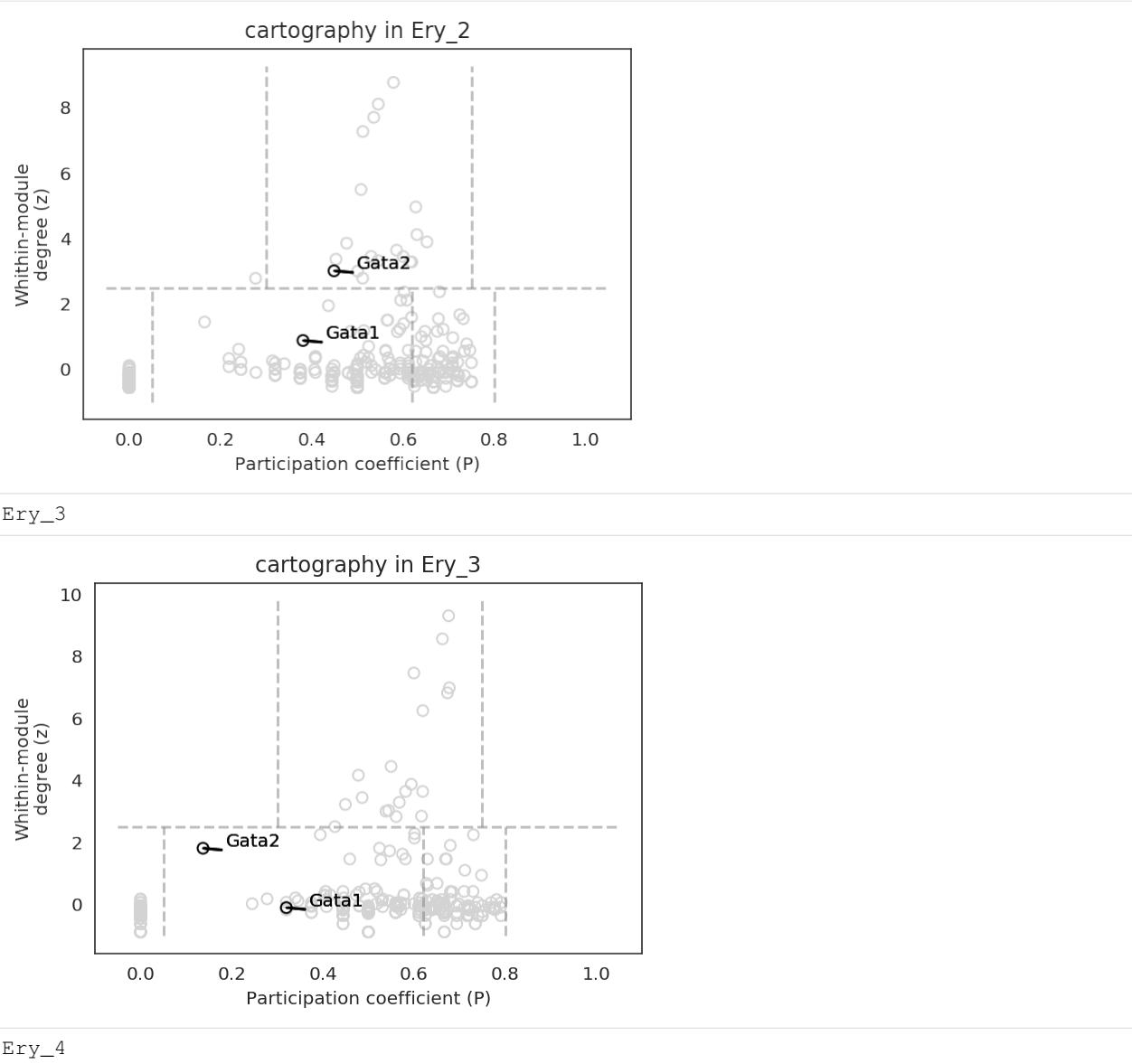
Ery_0

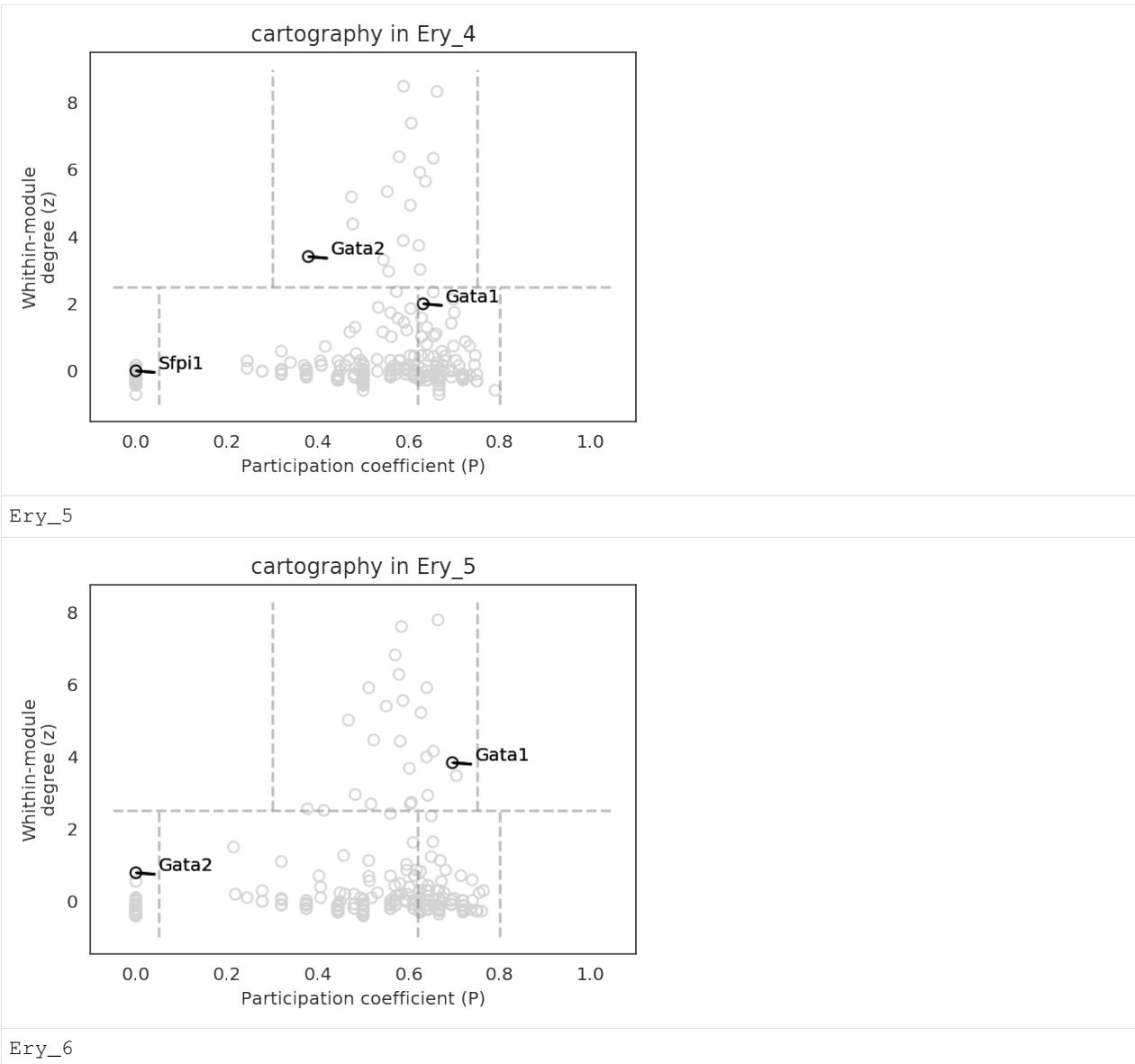


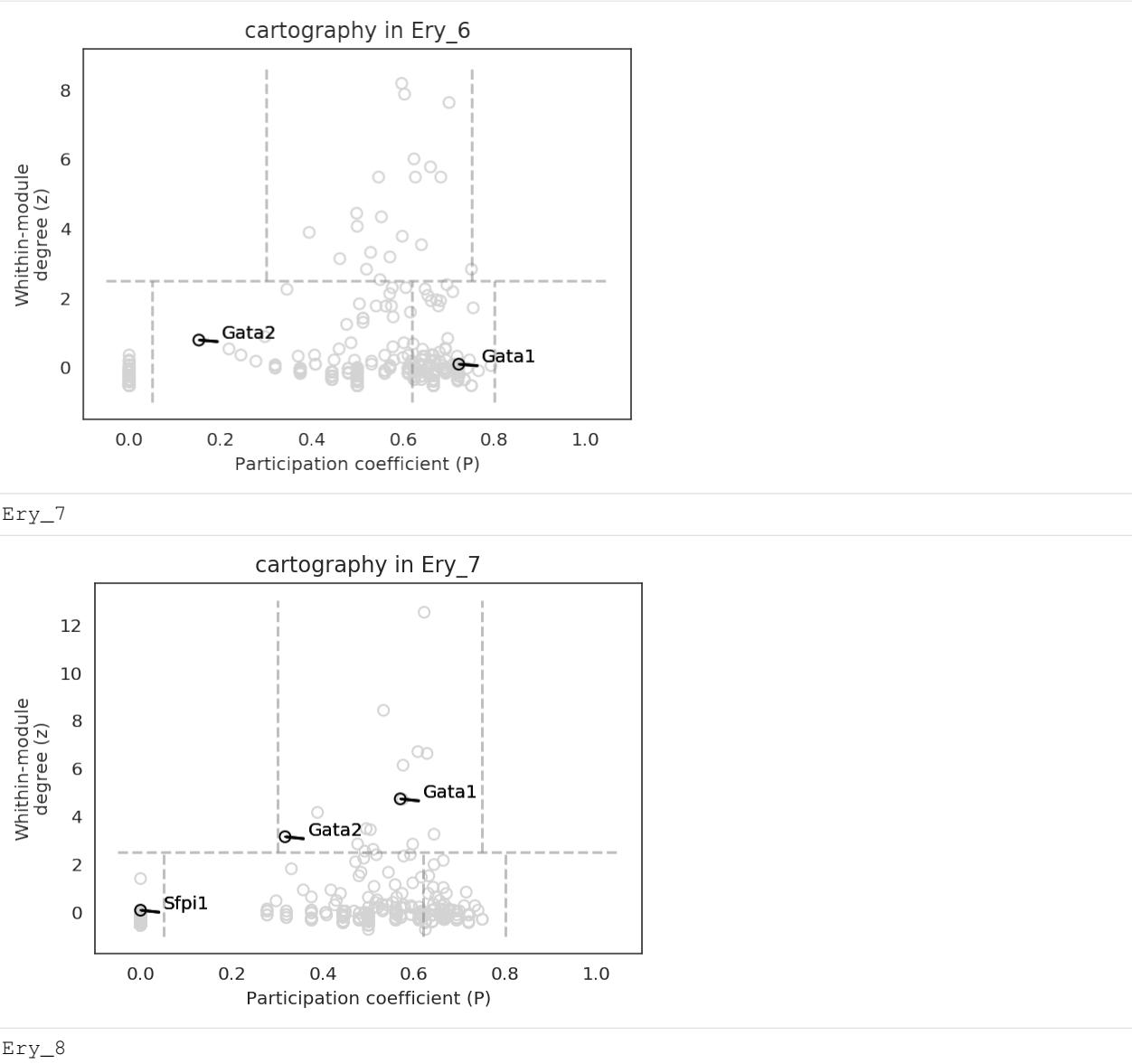
Ery_1

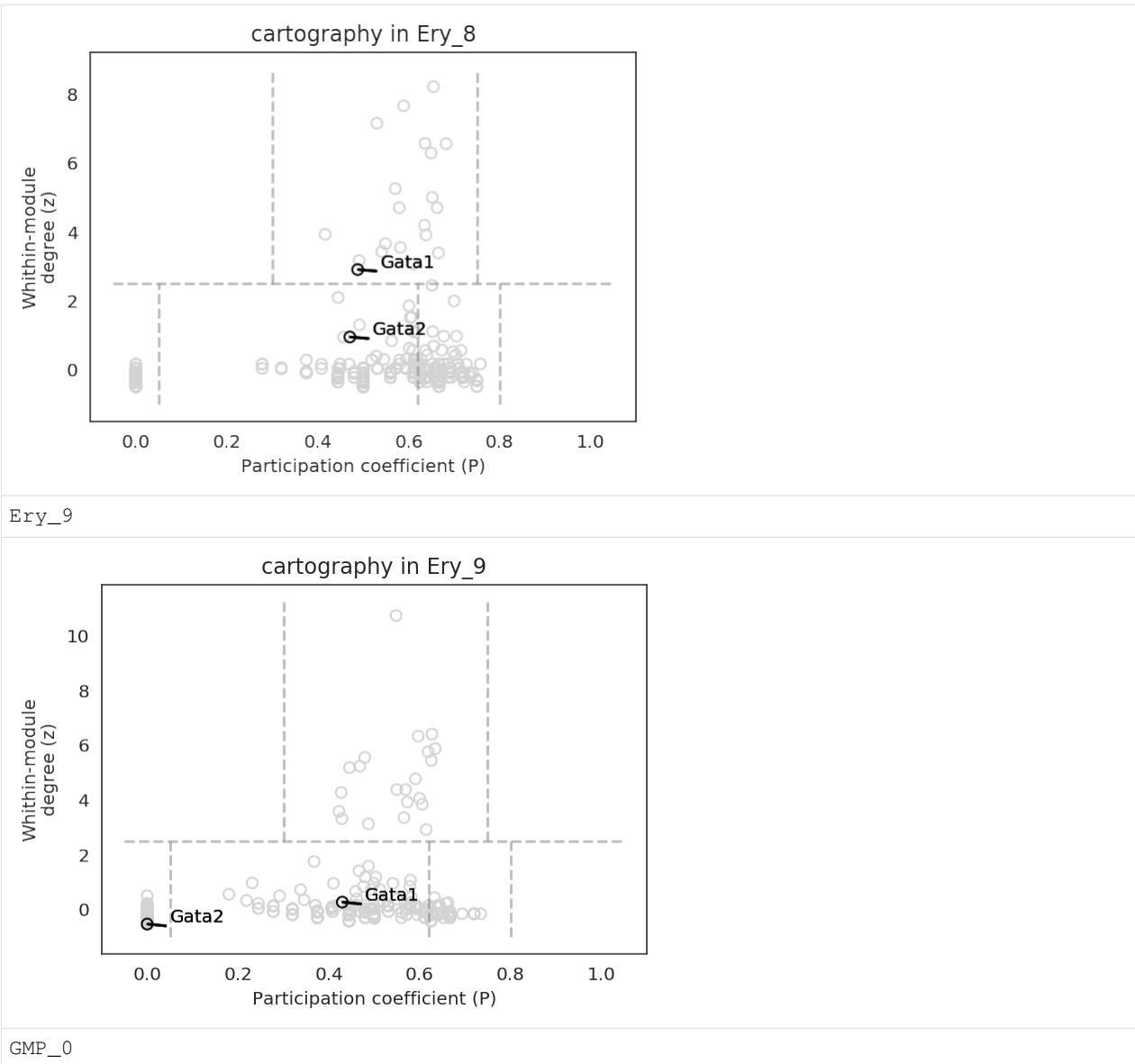


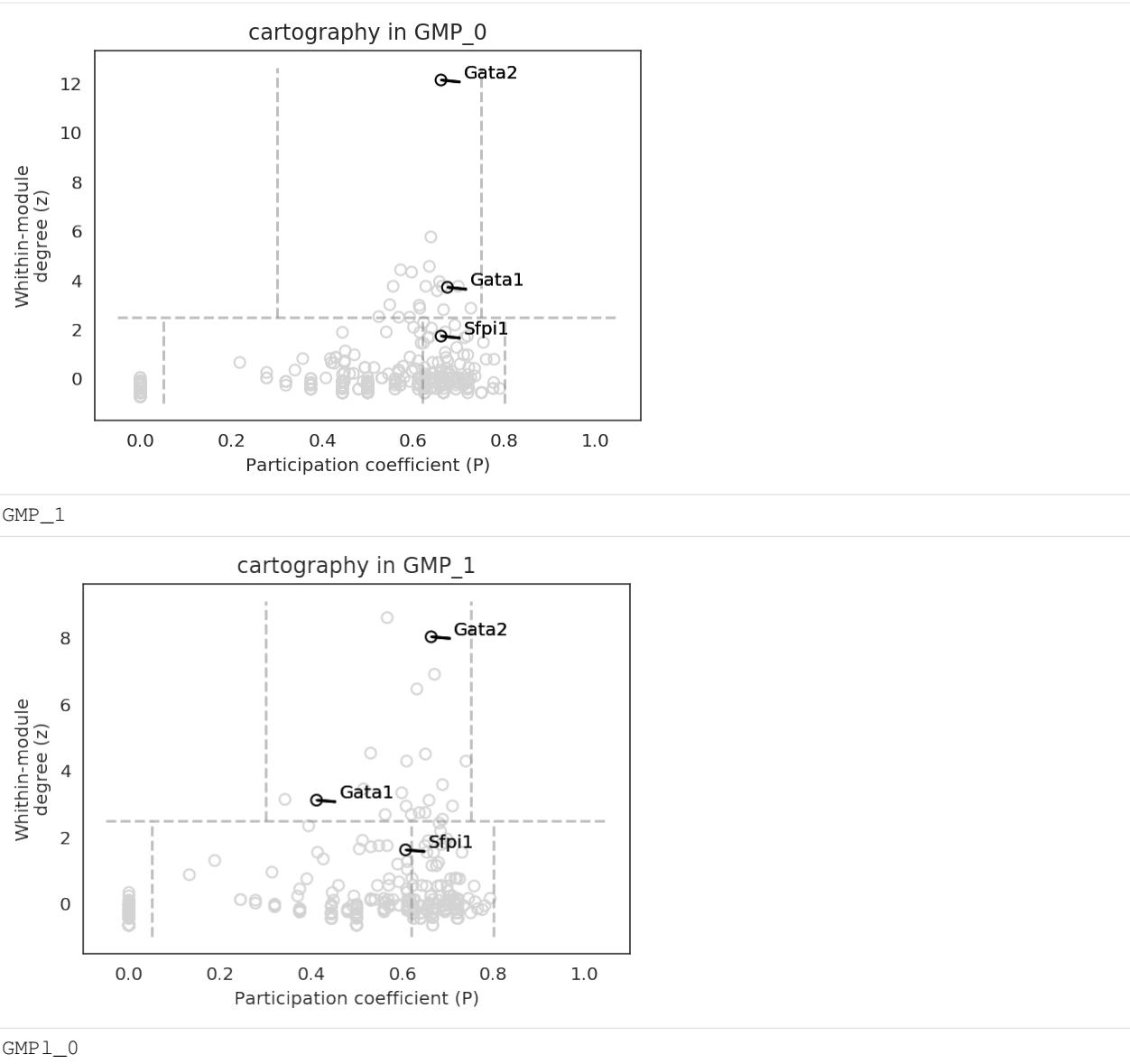
Ery_2

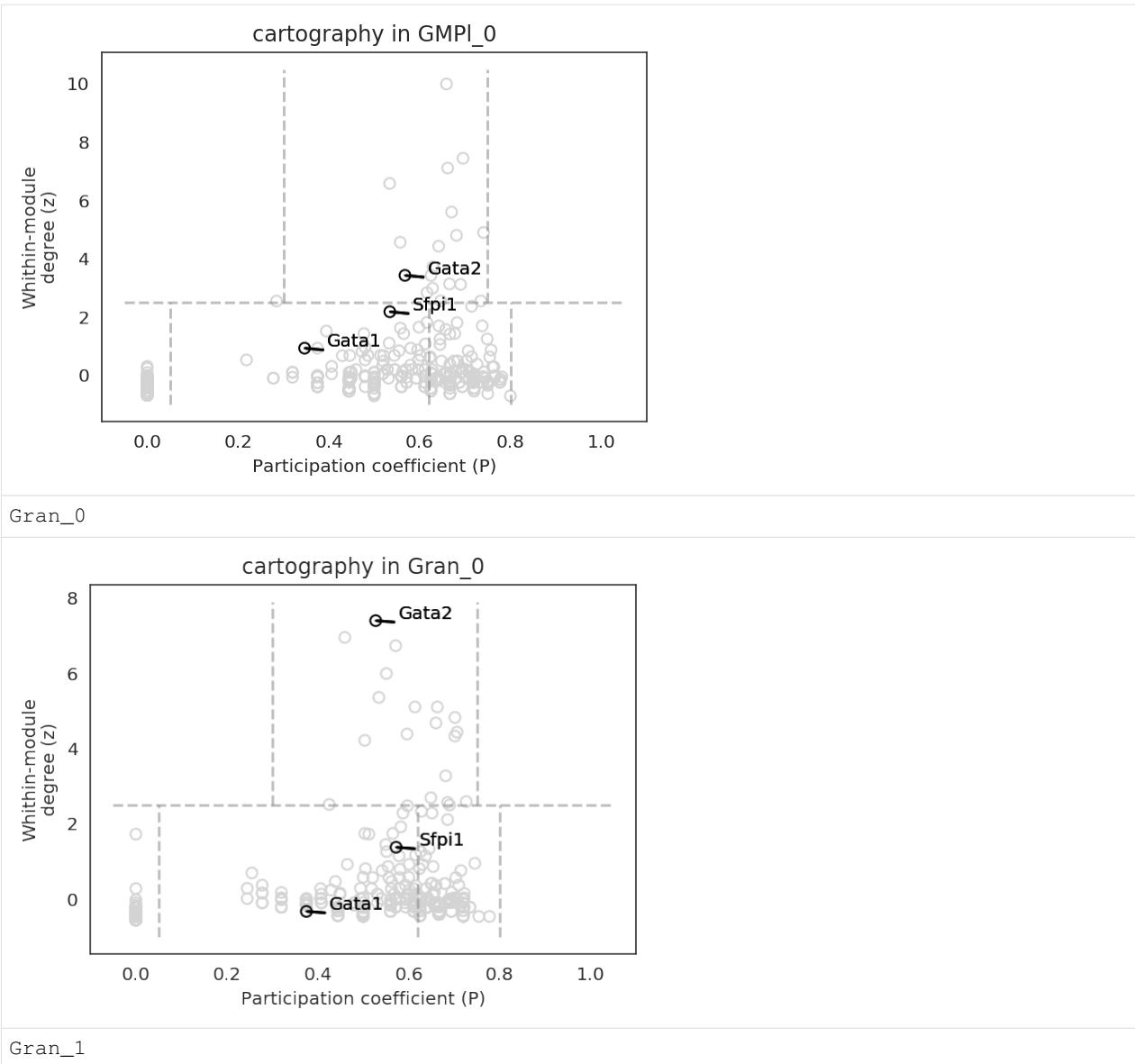


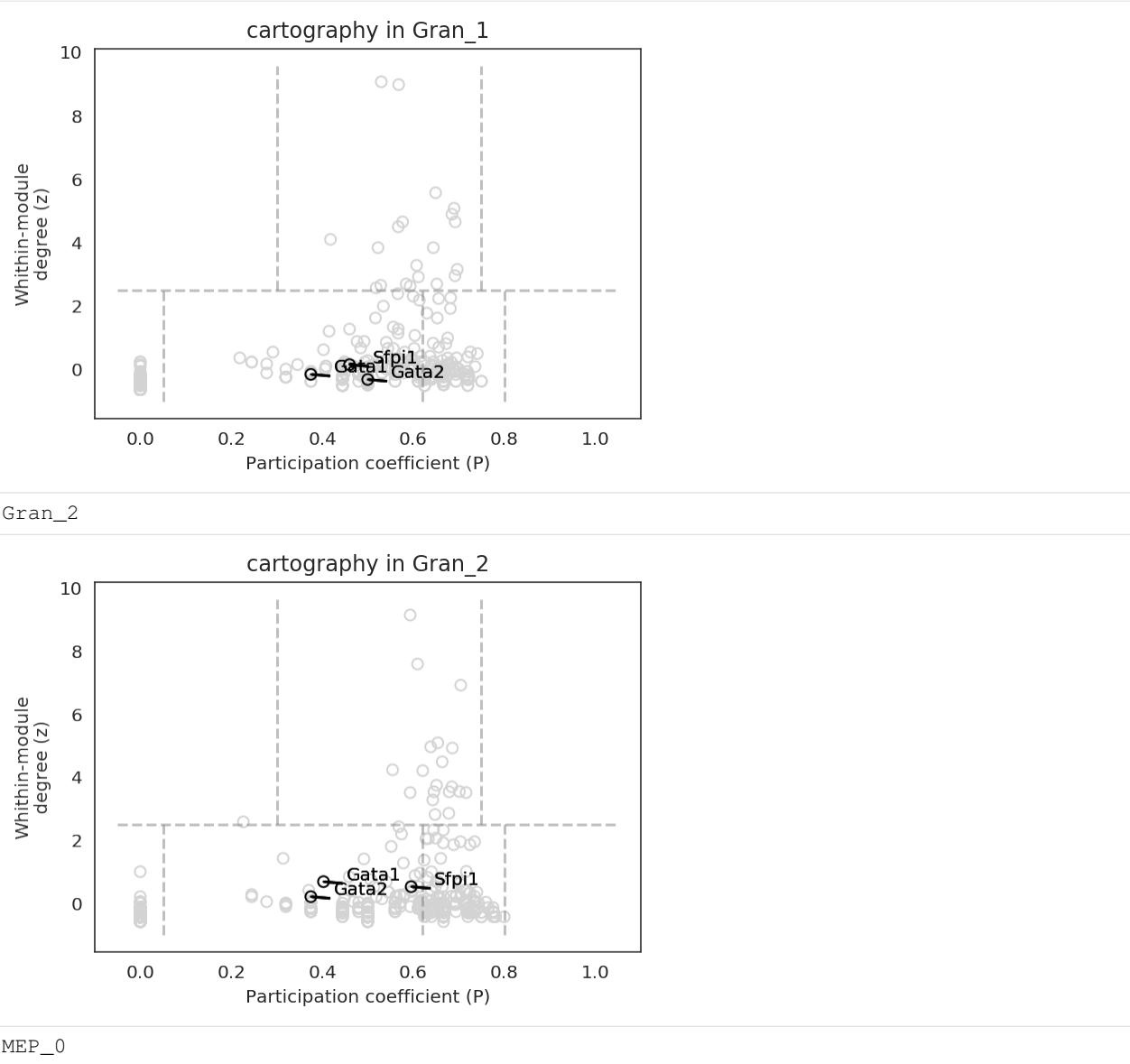


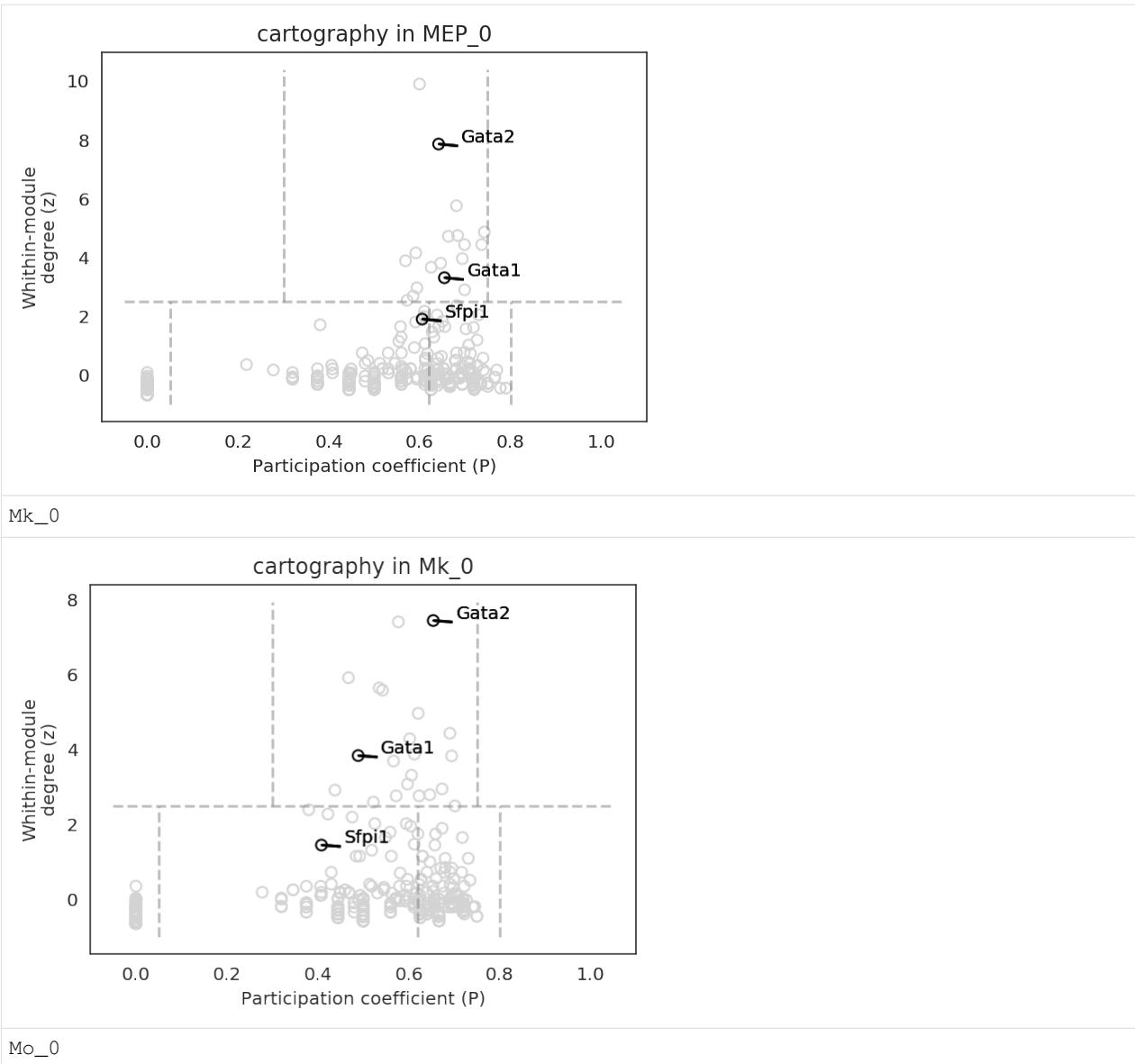


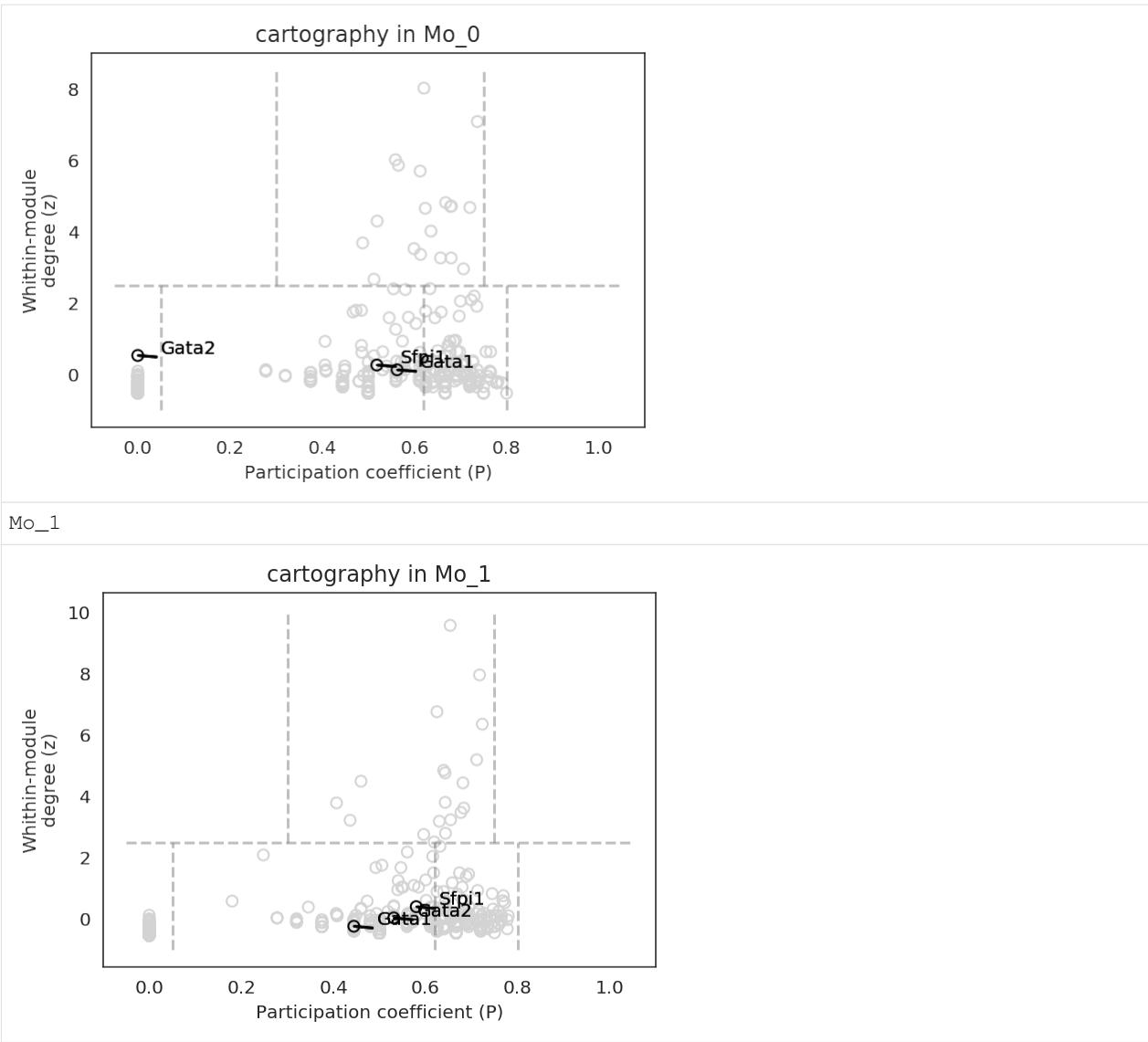








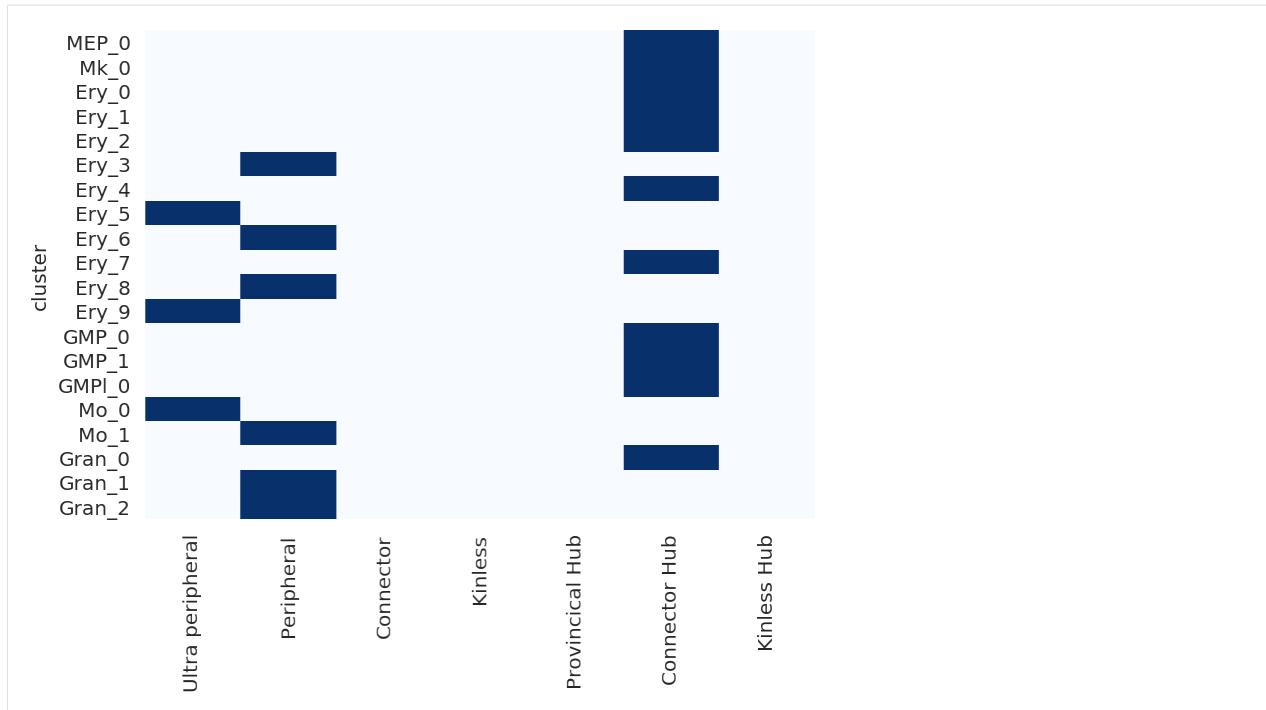




```
[66]: # Plot the summary of cartography analysis
```

```
links.plot_cartography_term(goi="Gata2", save=f"{save_folder}/cartography")
```

```
Gata2
```



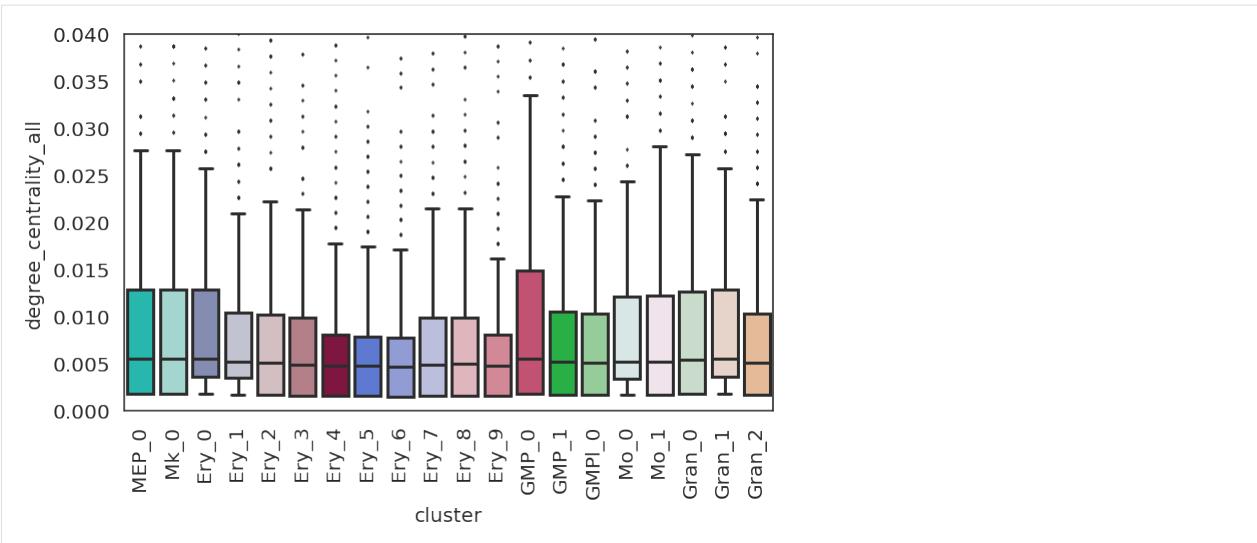
8. Network analysis; network score distribution

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

8.1. Distribution of network degree

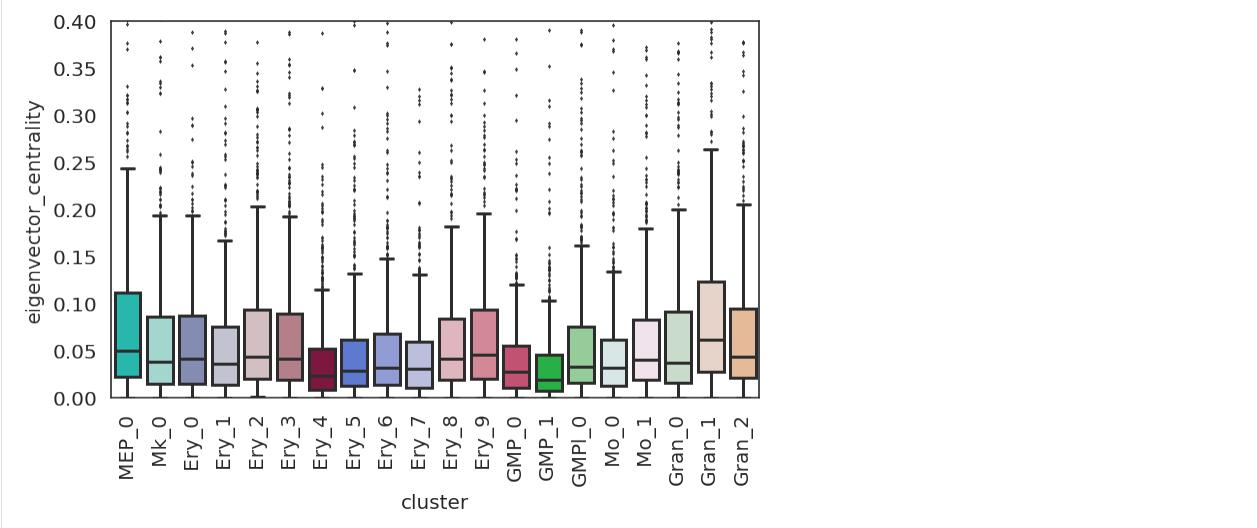
```
[60]: plt.subplots_adjust(left=0.15, bottom=0.3)
plt.ylim([0,0.040])
links.plot_score_distributions(values=["degree_centrality_all"], method="boxplot",
                                save=f"{save_folder}")
```

degree_centrality_all



```
[61]: plt.subplots_adjust(left=0.15, bottom=0.3)
plt.ylim([0, 0.40])
links.plot_score_distributions(values=["eigenvector_centrality"], method="boxplot", ↴
save=f'{save_folder}'")
```

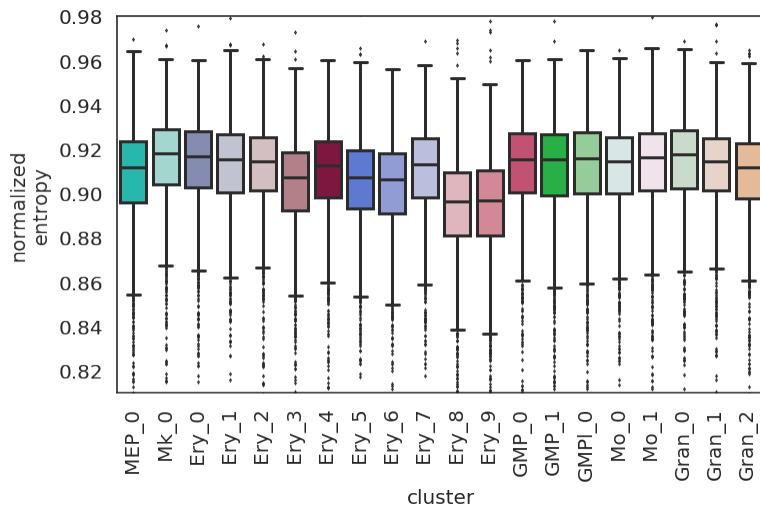
eigenvector_centrality



8.2. Distribution of netowrk entropy

```
[62]: plt.subplots_adjust(left=0.15, bottom=0.3)
links.plot_network_entropy_distributions(save=f"/{save_folder}/")
```

/home/k/anaconda3/envs/test/lib/python3.6/site-packages/scipy/stats/_distn_infrastructure.py:2614: RuntimeWarning: invalid value encountered in true_divide
pk = 1.0*pk / np.sum(pk, axis=0)
/home/k/anaconda3/envs/test/lib/python3.6/site-packages/celloracle/network_analysis/links_object.py:345: RuntimeWarning: divide by zero encountered in log
ent_norm.append(en/np.log(k[i]))
/home/k/anaconda3/envs/test/lib/python3.6/site-packages/celloracle/network_analysis/links_object.py:345: RuntimeWarning: invalid value encountered in double_scalars
ent_norm.append(en/np.log(k[i]))



Using the network scores, we could pick up cluster-specific key TFs. Gata2, Gata1, Klf1, E2f1, for example, are known to play an essential role in MEP, and these TFs showed high network score in our GRN.

However, it is important to note that network analysis alone cannot shed light on the specific functions or roles these TFs play in cell fate determination.

In the next section, we will begin to investigate each TF's contribution to cell fate by running GRN simulations

```
[ ]:
```

1.2.5 Simulation with GRNs

celloracle leverage GRNs to simulate signal propagation inside a cell. We can estimate the effect of gene perturbation by the simulation with GRNs.

Additionally, we will combine the signal propagation simulation with a cell state transition simulation. The latter simulation is performed by a python library for RNA-velocity analysis, called *velocyto*. This analysis may provide an insight into a complex system how TF controls enormous target genes to determines cell fate.

The jupyter notebook files and data used in this tutorial are available [here](#).

Python notebook

0. Import 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
```

```
[2]: import celloracle as co
```

```
[3]: 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

```
[5]: # Make folder to save plots
save_folder = "figures"
os.makedirs(save_folder, exist_ok=True)
```

1. Load data

1.1. Load processed oracle object

Load the oracle object. See the previous notebook for the notes on how to prepare the oracle object.

```
[7]: oracle = co.load_hdf5("../04_Network_analysis/Paul_15_data.celloracle.oracle")
```

1.2. Load inferred GRNs

In the previous notebook, we calculated GRNs. Now, we will use these GRNs for simulation. We import GRNs which were saved in the Links object.

```
[8]: links = co.load_hdf5("../04_Network_analysis/links.celloracle.links")
```

2. Make predictive models for simulation

We will fit ridge regression models again. This process takes less time than the GRN inference in the previous notebook because we only use significant TFs to predict target gene instead of 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 specific TF dict...
calculating GRN in Ery_0

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
models made for 1074 genes
calculating GRN in Ery_1

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
models made for 1092 genes
calculating GRN in Ery_2

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
models made for 1064 genes
calculating GRN in Ery_3

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
models made for 1105 genes
calculating GRN in Ery_4

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
models made for 1102 genes
calculating GRN in Ery_5

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
models made for 1116 genes
calculating GRN in Ery_6

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
models made for 1097 genes
calculating GRN in Ery_7

HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))

genes_in_gem: 1999
```

(continues on next page)

(continued from previous page)

```
models made for 1062 genes
calculating GRN in Ery_8
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1117 genes
calculating GRN in Ery_9
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1121 genes
calculating GRN in GMP_0
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1107 genes
calculating GRN in GMP_1
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1104 genes
calculating GRN in GMPl_0
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1089 genes
calculating GRN in Gran_0
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1067 genes
calculating GRN in Gran_1
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1076 genes
calculating GRN in Gran_2
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1105 genes
calculating GRN in MEP_0
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1152 genes
calculating GRN in Mk_0
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1114 genes
calculating GRN in Mo_0
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1085 genes
calculating GRN in Mo_1
```

```
HBox(children=(IntProgress(value=0, max=1999), HTML(value='')))
```

```
genes_in_gem: 1999
models made for 1074 genes
```

3. in silico Perturbation-simulation

Next, we will simulate the effects of perturbing a single TF to investigate its function and regulatory mechanism. See the celloracle paper for the details and scientific premise on the algorithm.

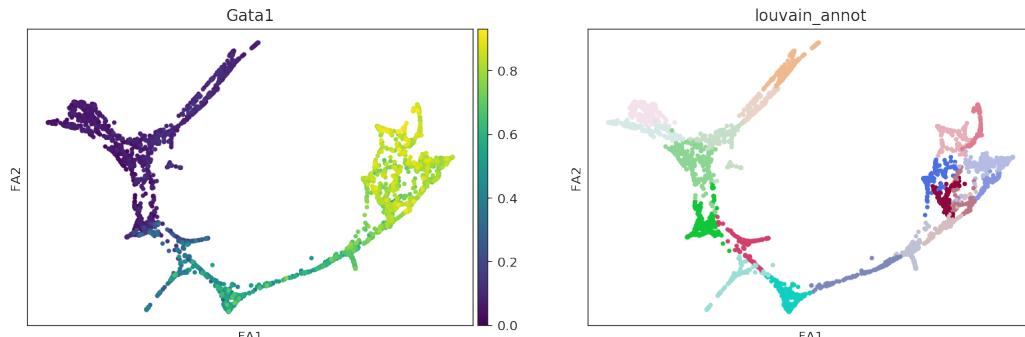
In this notebook, we'll show an example of the simulation; we'll simulate knock-out of Gata1 gene in the hematopoiesis.

Previous studies have shown that Gata1 is one of the TFs that regulates cell fate decisions in myeloid progenitors. Additionally, Gata1 has been shown to affect erythroid cell differentiation.

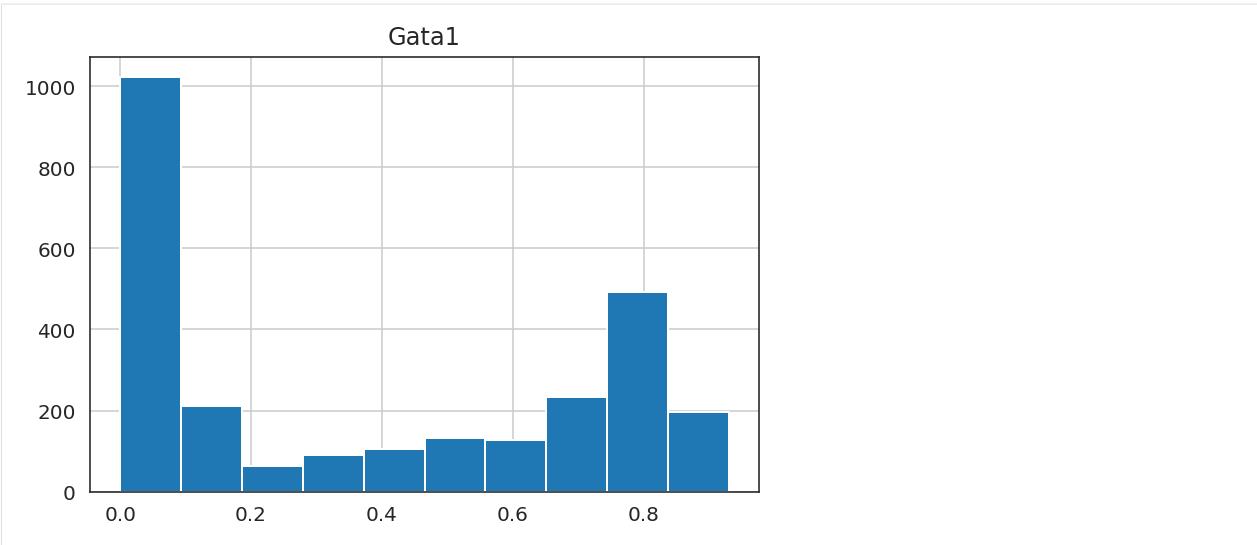
Here, we will analyze Gata1 for the demonstration of celloracle; Celloracle try to recapitulate the previous findings of Gata1 gene above.

3.1. 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")
```



```
[33]: # Plot gene expression in histogram
sc.get.obs_df(oracle.adata, keys=[goi], layer="imputed_count").hist()
plt.show()
```



3.2. calculate future gene expression after perturbation.

Although you can use any gene expression value for the input of in silico perturbation, we recommend avoiding extreme values which are far from natural gene expression ranges. If you set Gata1 gene expression to 100, for example, it may lead to biologically infeasible results.

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

```
[34]: # Enter perturbation conditions to simulate signal propagation after the perturbation.
oracle.simulate_shift(perturb_condition={goi: 0.0},
                      n_propagation=3)
```

3.3. calculate transition probability between cells

In the step above, we simulated simulated future gene expression values after perturbation. This prediction is based on iterative calculations of signal propagations within the GRN.

Next step, we will calculate the probability of a cell state transition based on the simulated data. Using the transition probability between cells, we can predict how a cell changes after perturbation.

This transition probability will be used in two ways.

- (1) Visualization of directed trajectory graph.
- (2) Markov simulation.

In Step 4.2 and 4.3, we use functions imported from the velocytoloom class in velocyto.py. Please see the documentation of VelocytoLoom for more information. http://velocyto.org/velocyto.py/fullapi/api_analysis.html

```
[35]: # Get transition probability
oracle.estimate_transition_prob(n_neighbors=200, knn_random=True, sampled_fraction=0.
                                ↵5)

# Calculate embedding
oracle.calculate_embedding_shift(sigma_corr = 0.05)
```

(continues on next page)

(continued from previous page)

```
# Calculate global trend of cell transition
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. Visualization

4.1. 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_kw_args=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_kw_args)

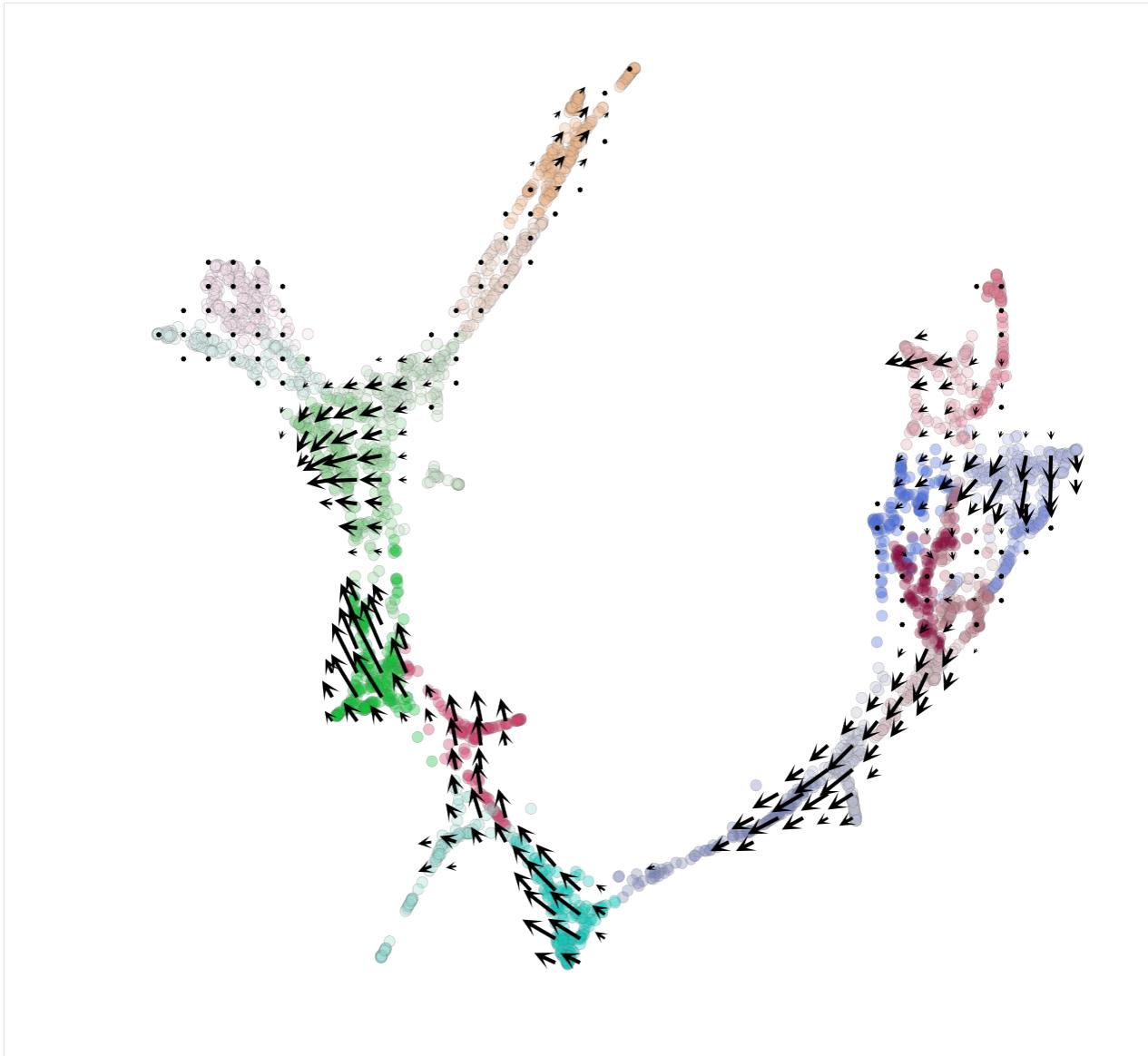
plt.axis("off")
# plt.savefig(f"{save_folder}/full_arrows(goi).png", transparent=True)

[36]: (-10815.27020913708, 10950.84121716522, -10711.36365432337, 10949.477199695968)
```



4.2. Grid graph

```
[37]: # Plot whole graph
plt.figure(None, (10,10))
oracle.plot_grid_arrows(quiver_scale=2.0,
                        scatter_kwarg_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)
```



5. Markov simulation to analyze the effects of perturbation on cell fate transition

We can also simulate cell state transition using Markov simulation.

5.1. Do Markov simulation

We will simulate using the parameters, “n_steps=200” and “n_duplication=5” in the following example.

To elaborate, this means:

- (1) We will do 200 times of iterative simulations to predict how the cell changes over time
- (2) We will repeat 5 rounds of simulations

```
[83]: %%time
# n_steps is the number of steps in markov simulation.
# n_duplication is the number of technical duplication 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
```

5.2. Check the results of the simulation for specific cells

Check the results of simulation. Pick up some cells and visualize their transition trajectory.

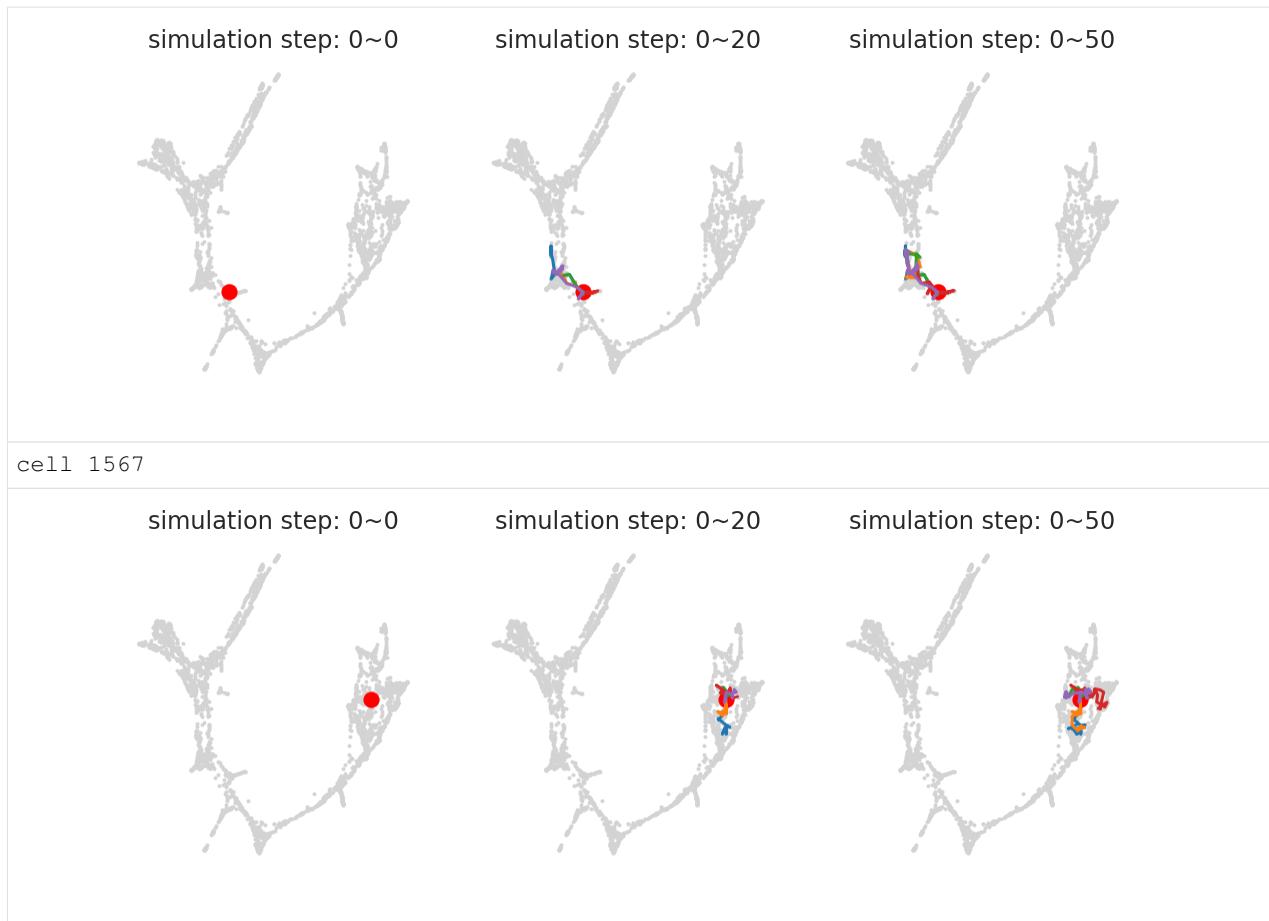
```
[88]: # Randomly pick up 3 cells
np.random.seed(12)
cells = oracle.adata.obs.index.values[np.random.choice(oracle.ixs_mcmc, 3)]

# 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
```



```
cell 43
```

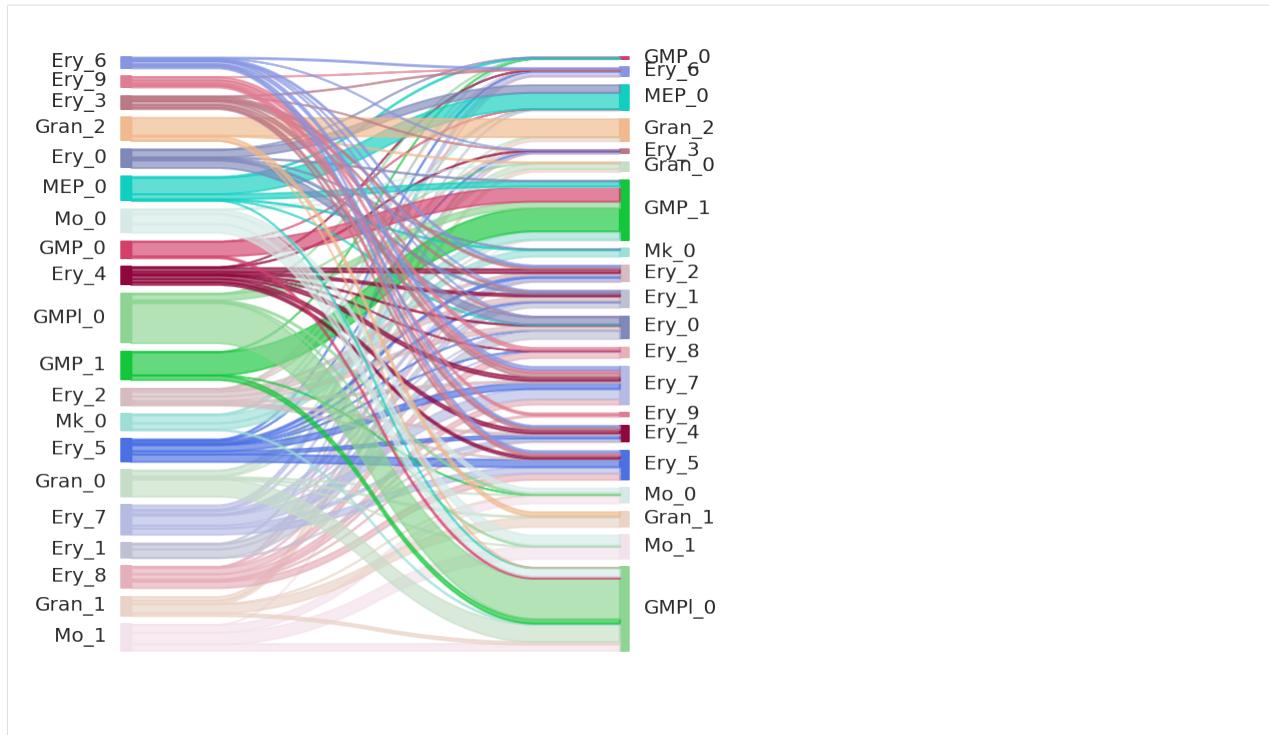


5.3. Summarize the results of simulation by plotting sankey diagram

Sankey diagrams are useful when you want to visualize proportional cell transitions between some groups.

For the grouping of cells, you can use arbitrary cluster unit.

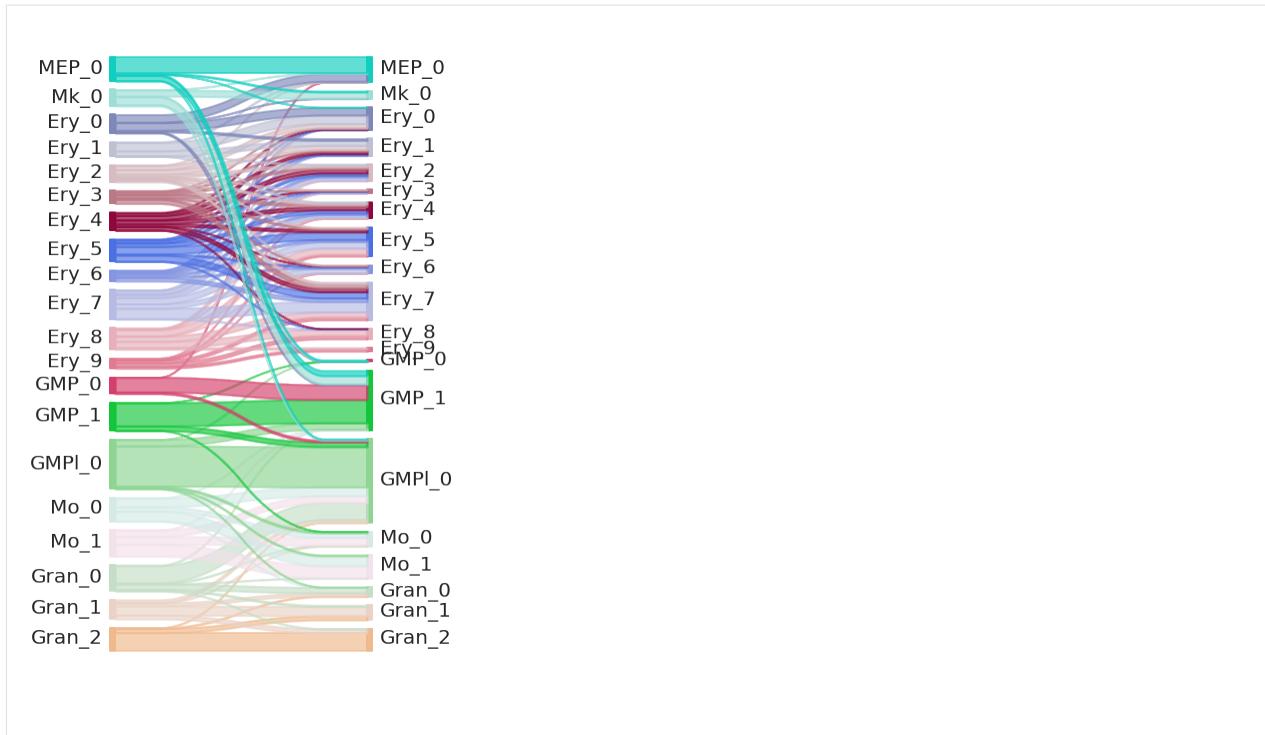
```
[89]: # Plot sankey diagram
plt.figure(figsize=[5, 6])
cl = "louvain_annot"
oracle.plot_mc_results_as_sankey(cluster_use=cl, start=0, end=100)
```



The Sankey diagram above looks messy because the cluster order is random.

Let's change the cluster order and make the plot again

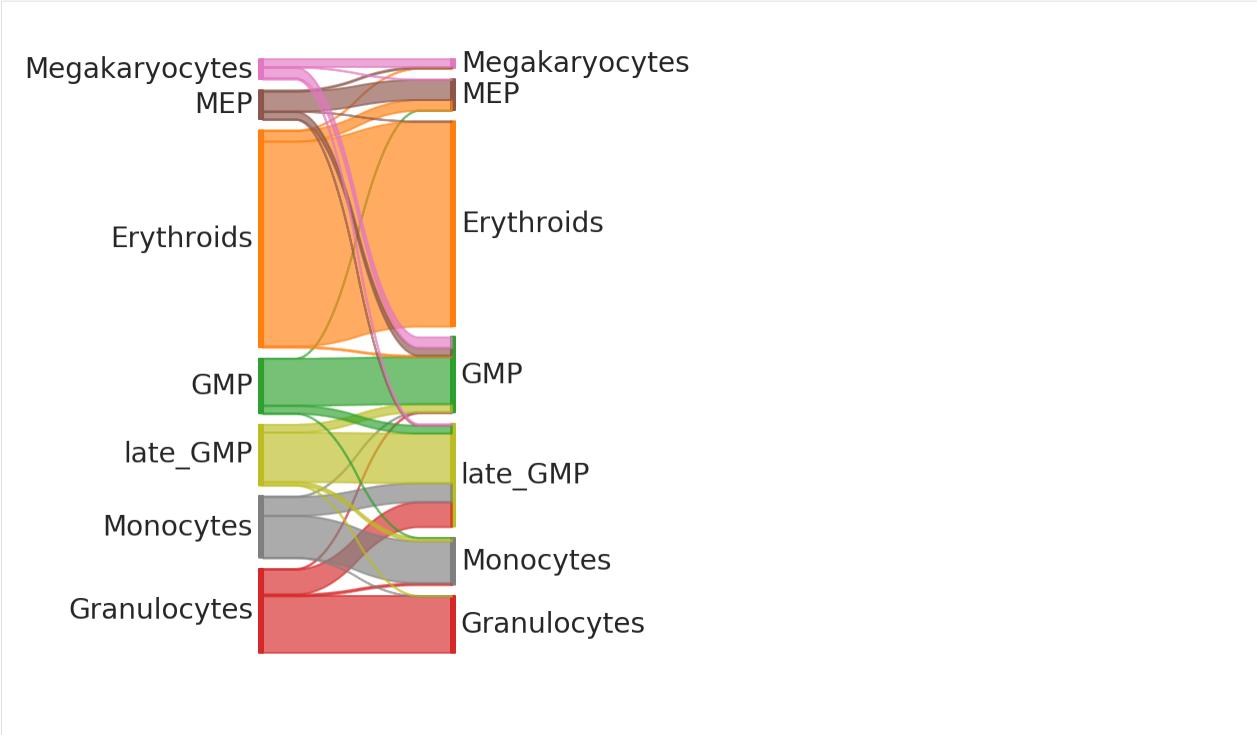
```
[90]: cl = "louvain_annot"
order = ['MEP_0', 'Mk_0', 'Ery_0', 'Ery_1', 'Ery_2', 'Ery_3', 'Ery_4',
         'Ery_5', 'Ery_6', 'Ery_7', 'Ery_8', 'Ery_9',
         'GMP_0', 'GMP_1', 'GMP_2', 'GMPI_0', 'GMPI_1',
         'Mo_0', 'Mo_1', 'Mo_2', 'Gran_0', 'Gran_1', 'Gran_2', 'Gran_3']
plt.figure(figsize=[5, 6])
plt.subplots_adjust(left=0.3, right=0.7)
oracle.plot_mc_results_as_sankey(cluster_use=cl, start=0, end=100, order=order)
# plt.savefig(f"{save_folder}/mcmc_{cl}.png")
```



Make another Saneky diagram with different cluster units.

```
[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_results_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)
```



Based on the results, we may conclude several things as follows.

Gata1 KO induced both cell state transitions from Erythroblasts to MEP, and from MEP to GMP.

- (1) These results suggest that Gata1 may play a role in the progression of Erythroid differentiation and cell state determination between the MEP and GMP lineages.
- (2) Gata1 KO also induced cell state transitions from granulocytes to late GMP, suggesting Gata1's involvement in Granulocytes differentiation.

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

1.3 API

1.3.1 Command Line API

CellOracle has a 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 object produced by this command can be used for input of celloracle.

```
seuratToAnndata YOUR_SEURAT_OBJECT.Rds OUTPUT_PATH
```

1.3.2 Python API

Custom class in celloracle

We define some custom classes in celloracle.

```
class celloracle.Links (name, links_dict={})  
    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

Dictionary that store unprocessed network data.

Type dictionary

filtered_links

Dictionary that store filtered network data.

Type dictionary

merged_score

Network scores.

Type pandas.dataframe

cluster

List of cluster name.

Type list of str

name

Name of clustering unit.

Type str

palette

DataFrame that store color information.

Type pandas.dataframe

```
filter_links (p=0.001, weight='coef_abs', thread_number=10000, genelist_source=None,  
genelist_target=None)
```

Filter network edges. In most cases, inferred GRN has non-significant random edges. We have to remove these edges before analyzing the network structure. You can do the filtering in any of the following ways.

- (1) Filter based on the p-value of the network edge. Please enter p-value for thresholding.
- (2) Filter 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 has several types of network weight, so you have to select which network weight do you want to use.
- (3) Filter 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'*)
Calculate network entropy scores.

Parameters **value** (*str*) – Default is “coef_abs”.

get_score (*test_mode=False*)

Get several network scores using R libraries. Make sure all dependent 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=True, kde=False, auto_gene_annot=False, percentile=98, args_dot={'n_levels': 105}, args_line={'c': 'gray'}, args_annot={}, save=None*)

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

Parameters

- **links** ([Links](#)) – See `network_analysis.Links` class for detail.
- **gois** (*list of str*) – 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*)

Plot the gene network cartography term 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](#)) – See `network_analysis.Links` class for detail.
- **gois** (*list of str*) – 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*)

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

Parameters

- **links** ([Links](#)) – See network_analysis.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*)

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

Parameters

- **links** (*Links object*) – See network_analysis.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*)

Make a scatter plot that compares specific network scores in two groups.

Parameters

- **links** ([Links](#)) – See network_analysis.Links class for detail.
- **value** (*srt*) – The network score type.
- **cluster1** (*str*) – Cluster name. Network scores in cluster1 will be visualized in the x-axis.
- **cluster2** (*str*) – Cluster name. Network scores in 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_distributions (*values=None, method='boxplot', save=None*)

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

Parameters

- **links** ([Links](#)) – See Links class for details.
- **values** (*list of str*) – The list of score to visualize. If it is None, all of the network score will be used.
- **method** (*str*) – Plotting method. Select either “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*)

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

Parameters

- **links** ([Links](#)) – See network_analysis.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*)

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

Parameters

- **links** ([Links](#)) – See network_analysis.Links class for detail.
- **cluster** (*str*) – Cluster name 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*)

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, cell_state=None, TFinfo_dic=None, annotation=None, verbose=True*)

Bases: `object`

Net is a custom class for inferring sample-specific GRN from scRNA-seq data. This class is used inside the Oracle class for GRN inference. This class requires two types of 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 the 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 (or base GRN): 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 derived from public ATAC-seq dataset of various tissue/cell type.

linkList

The results of the GRN inference.

Type `pandas.DataFrame`

all_genes

An array of all genes that exist in the input gene expression matrix

Type `numpy.array`

embedding_name

The key name name in `adata.obsm` containing dimensional reduction coordinates

Type `str`

annotation

Annotation. you can add custom annotation.

Type `dictionary`

coefs_dict

Coefs of linear regression.

Type dictionary

stats_dict
Statistic values about coefs.

Type dictionary

fitted_genes
List of genes where the regression model was successfully calculated.

Type list of str

failed_genes
List of genes that were not assigned coefs

Type list of str

cellstate
A metadata for GRN input

Type pandas.DataFrame

TFinfo
Information about potential regulatory TFs.

Type pandas.DataFrame

gem
Merged matrix made with gene_expression_matrix and cellstate matrix.

Type pandas.DataFrame

gem_standarized
Almost the same as gem, but the gene_expression_matrix was standarized.

Type pandas.DataFrame

library_last_update_date
Last update date of this code. This info is for code development. It can be deprecated in the future

Type str

object_initiation_date
The date when this object was made.

Type str

addAnnotation (annotation_dictionary)
Add a new annotation.

Parameters **annotation_dictionary** (*dictionary*) – e.g. {“sample_name”: “NIH 3T3 cell”}

addTFinfo_dictionary (TFdict)
Add a new TF info to pre-existing TFdict.

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

addTFinfo_matrix (TFinfo_matrix)
Load TF info dataframe.

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

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)

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). This calculation usually takes a long time.

Parameters

- **bagging_number** (*int*) – The number of estimators for bagging.
- **scaling** (*bool*) – Whether or not to scale regulatory gene expression values.
- **model_method** (*str*) – ML model name. Please select either "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 or not to show a progress bar.

fit_All_genes_parallel (*bagging_number*=200, *scaling*=True, *log*=None, *verbose*=10)

IMPORTANT: this function being debugged and is 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 or not 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)

Make ML models for genes of interest. This calculation will be performed in parallel using scikit-learn's bagging function. You can select a modeling method; Please chose either 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 or not to scale regulatory gene expression values.
- **model_method** (*str*) – ML model name. Please select either "bagging_ridge" or "bayesian_ridge"
- **save_coefs** (*bool*) – Whether or not 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 or not to show a progress bar.

plotCoefs (*target_gene*, *sort*=True, *threshold_p*=None)

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

Parameters

- **target_gene** (*str*) – gene name
- **sort** (*bool*) – Whether or not 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*)

Save object as hdf5.

Parameters **file_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.net’

updateLinkList (*verbose=True*)

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

Parameters **verbose** (*bool*) – Whether or not to show a progress bar

updateTFinfo_dictionary (*TFdict*)

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.

class `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 the future gene expression patterns and cell state transitions in response to the perturbation of TFs. Please see the CellOracle paper for details. The code of the Oracle class was made of the three components below.

- (1) Anndata: Gene expression matrix and metadata from single-cell RNA-seq are stored in the 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 processes several data to infer GRN. See the Net class documentation for details.
- (3) VelocytoLoom: Calculation of transition probability and visualization of directed trajectory graph will be performed in the same way as velocytoloom. VelocytoLoom is class from Velocyto, a python library for RNA-velocity analysis. In celloracle, we use some functions in velocytoloom for the visualization.

adata

Imported anndata object

Type anndata

cluster_column_name

The column name in adata.obs containing cluster info

Type str

embedding_name

The key name in adata.obsm containing dimensional reduction coordinates

Type str

addTFinfo_dictionary (*TFdict*)

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()

Deepcopy itself.

count_cells_in_mc_resutls (*cluster_use*, *end=-1*, *order=None*)

Count the simulated cell by the cluster.

Parameters

- **cluster_use** (*str*) – cluster information name in anndata.obs. You can use any cluster information in anndata.obs.
- **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.

Returns Number of cells before / after simulation

Return type pandas.DataFrame

fit_GRN_for_simulation (*GRN_unit='cluster'*, *alpha=1*, *use_cluster_specific_TFdict=False*)

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

GRN can be constructed for the entire population or each clusters. If you want to infer cluster-specific GRN, please set [*GRN_unit="cluster"*]. You can select cluster information when you import data.

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 increases, and you can detect weaker network connections. However, there may be more noise. If you select a higher value, it will reduce the chance of overfitting.

get_cluster_specific_TFdict_from_Links (*links_object*)

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`) – 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*)

Makes 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 increases, and you can detect weaker network connections. However, there may be more noise. If you select a higher value, it will reduce the chance of overfitting.
- **bagging_number** (*int*) – The number used in 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)
```

Load data about potential-regulatory TFs. You can import either TF_info_matrix or TFdict. For more information on how to make these files, please see the motif analysis module within the celloracle tutorial.

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)
```

Load scRNA-seq data. scRNA-seq data should be prepared as an anndata object. Preprocessing (cell and gene filtering, dimensional reduction, clustering, etc.) should be done before loading data. The method will import NORMALIZED and LOG TRANSFORMED data but NOT SCALED and NOT CENTERED data. See the tutorial for more details on how to process scRNA-seq data.

Parameters

- **adata** (`anndata`) – anndata object containing scRNA-seq data.
- **cluster_column_name** (`str`) – the name of column containing cluster information in `anndata.obs`. Clustering data should be in `anndata.obs`.
- **embedding_name** (`str`) – the key name for dimensional reduction information 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')
```

Load scRNA-seq data. scRNA-seq data should be prepared as an anndata object. Preprocessing (cell and gene filtering, dimensional reduction, clustering, 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 name of column containing cluster information in `anndata.obs`. Clustering data should be in `anndata.obs`.
- **embedding_name** (`str`) – the key name for dimensional reduction information 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={})
```

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 details (<https://seaborn.pydata.org/generated/seaborn.kdeplot.html#seaborn.kdeplot>).

plot_mc_result_as_trajectory(*cell_name*, *time_range*, *args*={})

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 list of index in Markov simulation
- **args** (*dictionary*) – dictionary for the arguments for matplotlib.pyplot.plot. See matplotlib documentation for details (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot).

plot_mc_results_as_sankey(*cluster_use*, *start*=0, *end*=-1, *order*=None, *font_size*=10)

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 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 the Sankey-diagram.
- **font_size** (*int*) – Font size for cluster name label in the Sankey diagram.

prepare_markov_simulation(*verbose*=False)

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, *calculation_randomized*=True)

Do Markov simulations 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 multiple times to get a robust outcome.

Parameters

- **n_steps** (*int*) – steps for Markov simulation. This value is equivalent to the amount of time after perturbation.
- **n_duplication** (*int*) – the number for multiple calculations.

simulate_shift(*perturb_condition*=None, *GRN_unit*='cluster', *n_propagation*=3, *ignore_warning*=False)

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

The simulation use three types of data. (1) GRN inference results (coef_matrix). (2) Perturb_condition: You can set arbitrary perturbation condition. (3) Gene expression matrix: The 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 infeasible values for the perturb condition. It is strongly recommended to check gene expression values in your data before selecting the perturb condition.

- **GRN_unit** (*str*) – GRN type. Please select either “whole” or “cluster”. See the documentation of “fit_GRN_for_simulation” for the detailed explanation.
- **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*, *random=False*)

This function summarizes the simulated cell state-transition by groping the results into each cluster. It returns summarized 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*)

Save object as hdf5.

Parameters **file_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.oracle’

updateTFinfo_dictionary (*TFdict*)

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

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

celloracle.load_hdf5 (*file_path*, *object_class_name=None*)

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

celloracle.motif_analysis module

The *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.

```
class celloracle.motif_analysis.TFinfo (peak_data_frame, ref_genome)
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 during GRN inference.

peak_df

dataframe about DNA peak and target gene data.

Type pandas.dataframe

all_target_gene
target genes.
Type array of str

ref_genome
reference genome name that was used in DNA peak generation.
Type str

scanned_df
Results of motif scan. Key is a peak name. Value is a dataframe of motif scan.
Type dictionary

dic_targetgene2TFs
Final product of motif scan. Key is a target gene. Value is a list of regulatory candidate genes.
Type dictionary

dic_peak2Targetgene
Dictionary. Key is a peak name. Value is a list of the target gene.
Type dictionary

dic_TF2targetgenes
Final product of motif scan. Key is a TF. Value is a list of potential target genes of the TF.
Type dictionary

copy()
Deepcopy itself.

filter_motifs_by_score(*threshold*, *method*=‘cumulative_score’)
Remove motifs with low binding scores.
Parameters **method** (str) – thresholding method. Select either of [“individual_score”, “cumulative_score”]

filter_peaks(*peaks_to_be_remainded*)
Filter peaks.
Parameters **peaks_to_be_remainded**(array of str) – list of peaks. Peaks that are NOT in the list will be removed.

make_TFinfo_dataframe_and_dictionary(*verbose*=True)
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()
Reset TF dictionary and TF dataframe. The following attributes will be erased: TF_onehot, dic_targetgene2TFs, dic_peak2Targetgene, dic_TF2targetgenes.

reset_filtering()
Reset filtering information. You can use this function to start 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)
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, *motifs*=None, *TF_evidence_level*=‘direct_and_indirect’)
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.
- **motifs** (*list*) – a list of gimmemotifs motifs, will revert to default_motifs() if None
- **TF_evidence_level** (*str*) – Please select one from [“direct”, “direct_and_indirect”]. If “direct” is selected, TFs that have a binding evidence were used. If “direct_and_indirect” is selected, TFs with binding evidence and inferred TFs are used. For more information, please read explanation of Motif class in gimmemotifs documentation (<https://gimmemotifs.readthedocs.io/en/master/index.html>)

to_dataframe (*verbose=True*)

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*)

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*)

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*)

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*)

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

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

`celloracle.motif_analysis.is_genome_installed(ref_genome)`

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.load_TFinfo(file_path)`

Load TFinfo object which was saved as hdf5 file.

Parameters `file_path` (`str`) – file path.

Returns Loaded TFinfo object.

Return type `TFinfo`

`celloracle.motif_analysis.load_TFinfo_from_parquets(folder_path)`

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`

`celloracle.motif_analysis.load_motifs(motifs_name)`

Load motifs from celloracle motif database

Parameters `motifs_name` (`str`) – Name of motifs.

Returns List of gimmemotifs.motif object.

Return type list

`celloracle.motif_analysis.make_TFinfo_from_scanned_file(path_to_raw_bed,`
`path_to_scanned_result_bed,`
`ref_genome)`

This function is currently an available.

`celloracle.motif_analysis.peak2fasta(peak_ids, ref_genome)`

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)`

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.remove_zero_seq(fasta_object)
```

Remove DNA sequence with zero length

```
celloracle.motif_analysis.scan_dna_for_motifs(scanner_object,          motifs_object, sequence_object,
                                              verbose=True)
```

This is a wrapper function to scan DNA sequences searchig for Gene motifs.

Parameters

- **scanner_object** (*gimmemotifs.scanner*) – Object that do motif scan.
- **motifs_object** (*gimmemotifs.motifs*) – Object that stores motif data.
- **sequence_object** (*gimmemotifs.fasta*) – Object that stores sequence data.

Returns scan results is stored in data frame.

Return type pandas.dataframe

Modules for Network analysis

celloracle.network_analysis module

The *network_analysis* module implements Network analysis.

```
class celloracle.network_analysis.Links(name, links_dict={})
```

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

Dictionary that store unprocessed network data.

Type dictionary

filtered_links

Dictionary that store filtered network data.

Type dictionary

merged_score

Network scores.

Type pandas.dataframe

cluster

List of cluster name.

Type list of str

name

Name of clustering unit.

Type str

palette

DataFrame that store color information.

Type pandas.dataframe

```
filter_links(p=0.001,           weight='coef_abs',           thread_number=10000,
```

```
genelist_source=None, genelist_target=None)
```

Filter network edges. In most cases, inferred GRN has non-significant random edges. We have

to remove these edges before analyzing the network structure. You can do the filtering in any of the following ways.

- (1) Filter based on the p-value of the network edge. Please enter p-value for thresholding.
- (2) Filter 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 has several types of network weight, so you have to select which network weight do you want to use.
- (3) Filter 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'*)

Calculate network entropy scores.

Parameters **value** (*str*) – Default is “coef_abs”.

get_score (*test_mode=False*)

Get several network scores using R libraries. Make sure all dependent 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=True, *kde=False*,
auto_gene_annot=False, *percentile=98*, *args_dot={'n_levels': 105}*,
args_line={'c': 'gray'},
args_annot={}, *save=None*)

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

Parameters

- **links** ([Links](#)) – See `network_analysis.Links` class for detail.
- **gois** (*list of str*) – 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*)

Plot the gene network cartography term 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](#)) – See `network_analysis.Links` class for detail.

- **gois** (*list of str*) – 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*)

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

Parameters

- **links** (*Links*) – See network_analysis.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*)

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

Parameters

- **links** (*Links object*) – See network_analysis.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*)

Make a scatter plot that compares specific network scores in two groups.

Parameters

- **links** (*Links*) – See network_analysis.Links class for detail.
- **value** (*srt*) – The network score type.
- **cluster1** (*str*) – Cluster name. Network scores in cluster1 will be visualized in the x-axis.
- **cluster2** (*str*) – Cluster name. Network scores in 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_distributions (*values=None, method='boxplot', save=None*)

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

Parameters

- **links** (*Links*) – See Links class for details.
- **values** (*list of str*) – The list of score to visualize. If it is None, all of the network score will be used.
- **method** (*str*) – Plotting method. Select either “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*)

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

Parameters

- **links** ([Links](#)) – See network_analysis.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*)

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

Parameters

- **links** ([Links](#)) – See network_analysis.Links class for detail.
- **cluster** (*str*) – Cluster name 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*)

Save object as hdf5.

Parameters **file_path** (*str*) – file path to save file. Filename needs to end with ‘.celloracle.links’

`celloracle.network_analysis.draw_network(linkList, return_graph=False)`

Plot network graph.

Parameters

- **linkList** ([pandas.DataFrame](#)) – GRN saved as linkList.
- **return_graph** (*bool*) – Whether to return graph object.

Returns Network X graph object.

Return type Graph object

`celloracle.network_analysis.get_R_path()`

`celloracle.network_analysis.get_links(oracle_object, cluster_name_for_GRN_unit=None, alpha=10, bagging_number=20, verbose_level=1, test_mode=False)`

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](#)) – 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 increases, and you can detect weaker network connections. However, there may be more noise. If you select a higher value, it will reduce the chance of overfitting.
- **bagging_number** (*int*) – The number used in 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.linkList_to_networkgraph(filteredlinkList)`
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.load_links(file_path)`
Load links object saved as a hdf5 file.

Parameters `file_path` (*str*) – file path.

Returns loaded links object.

Return type `Links`

`celloracle.network_analysis.set_R_path(R_path)`

`celloracle.network_analysis.test_R_libraries_installation(show_all_stdout=False)`
CellOracle.network_analysis use several R libraries for network analysis. This is a test function to check for instalation of the necessary R libraries.

`celloracle.network_analysis.transfer_scores_from_links_to_adata(adata,
links,
method='median')`

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

Parameters

- **adata** (*anndata*) – anndata
- **links** (`Links`) – Likns object
- **method** (*str*) – The method to summarize data.

Other modules

`celloracle.go_analysis module`

The `go_analysis` module implements Gene Ontology analysis. This module use goatools internally.

`celloracle.go_analysis.geneID2Symbol(IDs, species='mouse')`
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.geneSymbol2ID(symbols, species='mouse')`
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.get_GO(gene_query, species='mouse')
```

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

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

```
celloracle.utility.exec_process(commands, message=True, wait_finished=True, turn_process=True)
```

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 or not to wait for the process to finish. If false, the process will be perfomed in background and the function will finish immediately
- **return_process** (*bool*) – Whether to return “process”.

```
celloracle.utility.intersect(list1, list2)
```

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.knn_data_transferer(adata_ref, adata_que, n_neighbours=20, cluster_name=None, embedding_name=None, adata_true=None, transfer_color=True, n_PCA=30, use_PCA_in_adata=False)
```

Extract categorical information from adata.obs or embedding information from adata.obsm and transfer these information into query anndata. In the calculation, KNN is used after PCA.

Parameters

- **adata_ref** (*anndata*) – reference anndata
- **adata_que** (*anndata*) – query anndata
- **cluster_name** (*str or list of str*) – cluster name(s) to be transferred. If you want to transfer multiple data, you can set the cluster names as a list.
- **embedding_name** (*str or list of str*) – embedding name(s) to be transferred. If you want to transfer multiple data, you can set the embed-

- ding names as a list.
- **adata_true** (*str*) – This argument can be used for the validation of this algorithm. If you have true answer, you can set it. If you set true answer, the function will return some metrics for benchmarking.
 - **transfer_color** (*bool*) – Whether or not to transfer color data in addition to cluster information.
 - **n_PCA** (*int*) – Number of PCs that will be used for the input of KNN algorithm.

```
celloracle.utility.load_hdf5(file_path, object_class_name=None)
```

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.load_pickled_object(filepath)
```

Load pickled object.

Parameters **filepath** (*str*) – file path.

Returns loaded object.

Return type python object

```
class celloracle.utility.makelog(file_name=None, directory=None)
```

Bases: object

This is a class for making log.

info (*comment*)

Add comment into the log file.

Parameters **comment** (*str*) – comment.

```
celloracle.utility.save_as_pickled_object(obj,filepath)
```

Save any object using pickle.

Parameters

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

```
celloracle.utility.standard(df)
```

Standardize value.

Parameters **df** (*pandas.DataFrame*) – dataframe.

Returns Data after standardization.

Return type pandas.DataFrame

```
celloracle.utility.transfer_all_colors_between_anndata(adata_ref,
                                                       adata_que)
```

Extract all color information from reference anndata and transfer the color into query anndata.

Parameters

- **adata_ref** (*anndata*) – reference anndata
- **adata_que** (*anndata*) – query anndata

```
celloracle.utility.transfer_color_between_anndata(adata_ref,
                                                 adata_que,
                                                 clus-
                                                 ter_name)
```

Extract color information from reference anndata and transfer the color into query anndata.

Parameters

- **adata_ref** (*anndata*) – reference anndata
- **adata_que** (*anndata*) – query anndata

- **cluster_name** (*str*) – cluster name. This information should exist in the anndata.obs.

```
celloracle.utility.update_adata(adata)
```

celloracle.data module

The `data` module implements data download and loading.

```
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

The `data_conversion` module implements data conversion between different platform.

```
celloracle.data_conversion.seurat_object_to_anndata(file_path_seurat_object,  
                                                 delete_tmp_file=True)
```

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

1.4 Changelog

• 0.5.1 <2020-08-4>

- Add new promoter-TSS reference data for several reference genomes; (1)"Xenopus": ["xenTro2", "xenTro3"], (2)"Rat": ["rn4", "rn5", "rn6"], (3)"Drosophila": ["dm3", "dm6"], (4)"C.elegans": ["ce6", "ce10"], (5)"Arabidopsis": ["tair10"].

- Add new motif data for several species: "Xenopus", "Rat", "Drosophila", "C.elegans" and "Arabidopsis".

• 0.5.0 <2020-08-3>

- Add now functions for custom motifs. You can select motifs from several options. Also, we updated our web tutorial to introduce how to load / make a different motif data.

- Change default motifs for S.cerevisiae and Zebrafish.

- Change requirements for dependent package: gimmermotifs and geomepy. Celloracle codes were updated to support new version of gimmermotifs (0.14.4) and geomepy (0.8.4).

• 0.4.2 <2020-07-14>

- Add promoter-TSS information for Zebrafish reference genome (danRer7, danRer10 and danRer11).

• 0.4.1 <2020-07-02>

- Add promoter-TSS information for S.cerevisiae reference genome (sacCer2 and sacCer3).
- 0.4.0 <2020-06-28>
 - Change requirements.
 - From this version, pandas version 1.0.3 or later is required.
 - From this version, scanpy version 1.5.3 or later is required.
- 0.3.7 <2020-06-12>
 - Delete GO function from r-script
 - Update some functions for network visualization
- 0.3.6 <2020-06-08>
 - Fix a bug on the transition probability calculation in Markov simulation
 - Add new function “count_cells_in_mc_results” to oracle class
- 0.3.5 <2020-05-09>
 - Fix a bug on the function for gene cortography visualization
 - Change some settings for installation
 - Update data conversion module
- 0.3.4 <2020-04-29>
 - Change pandas version restriction
 - Fix a bug on the function for gene cortography visualization
 - Add new functions for R-path configuration
- 0.3.3 <2020-04-24>
 - Add promoter-TSS information for hg19 and hg38 reference genome
- 0.3.1 <2020-03-23>
 - Fix an error when try to save file larger than 4GB file
- 0.3.0 <2020-2-17>
 - Release beta version

1.5 License

The software is provided under a modified Apache License Version 2.0. The software may be used for non-commercial academic purposes only. For any other use of the Work, including commercial use, please contact Morris lab.

Copyright 2020 Kenji Kamimoto, Christy Hoffmann, Samantha Morris

Apache License Version 2.0, January 2004 <http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

“License” shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

“Licensor” shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

“Legal Entity” shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, “control” means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

“You” (or “Your”) shall mean an individual or Legal Entity exercising permissions granted by this License.

“Source” form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

“Object” form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

“Work” shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

“Derivative Works” shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

“Contribution” shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licenser for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, “submitted” means any form of electronic, verbal, or written communication sent to the Licenser or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licenser for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as “Not a Contribution.”

“Contributor” shall mean Licenser and any individual or Legal Entity on behalf of whom a Contribution has been received by Licenser and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a “NOTICE” text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices

that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets “[]” replaced with your own identifying information. (Don’t include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same “printed page” as the copyright notice for easier identification within third-party archives.

Copyright 2020 Kenji Kamimoto, Christy Hoffmann, Samantha Morris

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1.6 Authors and citations

1.6.1 Cite celloracle

If you use celloracle please cite our bioarxiv preprint [CellOracle: Dissecting cell identity via network inference and in silico gene perturbation](#).

1.6.2 celloracle software development

celloracle is developed and maintained by Kenji Kamimoto and members of Samantha Morris Lab. Please post troubles or questions on the [Github repository](#).

**CHAPTER
TWO**

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

`celloracle`, 115
`celloracle.data`, 136
`celloracle.data_conversion`, 136
`celloracle.go_analysis`, 133
`celloracle.motif_analysis`, 125
`celloracle.network_analysis`, 129
`celloracle.utility`, 134

INDEX

A

adata (*celloracle.Oracle attribute*), 121
addAnnotation() (*celloracle.Net method*), 119
addTFinfo_dictionary() (*celloracle.Net method*), 119
addTFinfo_dictionary() (*celloracle.Oracle method*), 121
addTFinfo_matrix() (*celloracle.Net method*), 119
all_genes (*celloracle.Net attribute*), 118
all_target_gene (*celloracle.motif_analysis.TFinfo attribute*), 126
annotation (*celloracle.Net attribute*), 118

C

celloracle
 module, 115
celloracle.data
 module, 136
celloracle.data_conversion
 module, 136
celloracle.go_analysis
 module, 133
celloracle.motif_analysis
 module, 125
celloracle.network_analysis
 module, 129
celloracle.utility
 module, 134
cellstate (*celloracle.Net attribute*), 119
cluster (*celloracle.Links attribute*), 115
cluster (*celloracle.network_analysis.Links attribute*), 129
cluster_column_name (*celloracle.Oracle attribute*), 121
coefs_dict (*celloracle.Net attribute*), 118
copy () (*celloracle.motif_analysis.TFinfo method*), 126
copy () (*celloracle.Net method*), 119
copy () (*celloracle.Oracle method*), 122
count_cells_in_mc_resutls() (*celloracle.Oracle method*), 122

D

dic_peak2Targetgene (*celloracle.motif_analysis.TFinfo attribute*), 126
dic_targetgene2TFs (*celloracle.motif_analysis.TFinfo attribute*), 126
dic_TF2targetgenes (*celloracle.motif_analysis.TFinfo attribute*), 126
draw_network() (*in module celloracle.network_analysis*), 132

E

embedding_name (*celloracle.Net attribute*), 118
embedding_name (*celloracle.Oracle attribute*), 121
exec_process () (*in module celloracle.utility*), 134

F

failed_genes (*celloracle.Net attribute*), 119
filter_links() (*celloracle.Links method*), 115
filter_links() (*celloracle.network_analysis.Links method*), 129
filter_motifs_by_score() (*celloracle.motif_analysis.TFinfo method*), 126
filter_peaks() (*celloracle.motif_analysis.TFinfo method*), 126
filtered_links (*celloracle.Links attribute*), 115
filtered_links (*celloracle.network_analysis.Links attribute*), 129
fit_All_genes() (*celloracle.Net method*), 119
fit_All_genes_parallel() (*celloracle.Net method*), 120
fit_genes() (*celloracle.Net method*), 120
fit_GRN_for_simulation() (*celloracle.Oracle method*), 122
fitted_genes (*celloracle.Net attribute*), 119

G

gem (*celloracle.Net attribute*), 119
gem_standerdized (*celloracle.Net attribute*), 119
geneID2Symbol() (*in module celloracle.go_analysis*), 133
geneSymbol2ID() (*in module celloracle.go_analysis*), 133

get_cluster_specific_TFdict_from_Links() (in module `celloracle`.*Oracle method*), 122
get_GO() (in module `celloracle`.*go_analysis*), 134
get_links() (in module `celloracle`.*Oracle method*), 122
get_links() (in module `celloracle`.*network_analysis*), 132
get_network_entropy() (in module `celloracle`.*Links method*), 116
get_network_entropy() (in module `celloracle`.*network_analysis.Links method*), 130
get_R_path() (in module `celloracle`.*network_analysis*), 132
get_score() (in module `celloracle`.*Links method*), 116
get_score() (in module `celloracle`.*network_analysis.Links method*), 130
get_tss_info() (in module `celloracle`.*motif_analysis*), 127

|

import_anndata_as_normalized_count() (in module `celloracle`.*Oracle method*), 123
import_anndata_as_raw_count() (in module `celloracle`.*Oracle method*), 123
import_TF_data() (in module `celloracle`.*Oracle method*), 122
info() (in module `celloracle`.*makelog* method), 135
integrate_tss_peak_with_cicero() (in module `celloracle`.*motif_analysis*), 127
intersect() (in module `celloracle`.*utility*), 134
is_genome_installed() (in module `celloracle`.*motif_analysis*), 128

K

knn_data_transferer() (in module `celloracle`.*utility*), 134

L

library_last_update_date (in module `celloracle`.*Net attribute*), 119
linkList (in module `celloracle`.*Net attribute*), 118
linkList_to_networkgraph() (in module `celloracle`.*network_analysis*), 133
Links (class in `celloracle`), 115
Links (class in `celloracle`.*network_analysis*), 129
links_dict (in module `celloracle`.*Links attribute*), 115
links_dict (in module `celloracle`.*network_analysis.Links attribute*), 129
load_hdf5() (in module `celloracle`), 125
load_hdf5() (in module `celloracle`.*utility*), 135
load_links() (in module `celloracle`.*network_analysis*), 133
load_motifs() (in module `celloracle`.*motif_analysis*), 128
load_pickled_object() (in module `celloracle`.*utility*), 135

get_cluster_specific_TFdict_from_Links() (in module `celloracle`.*motif_analysis*), 128
load_TFinfo_df_mm9_mouse_atac_atlas() (in module `celloracle`.*data*), 136
load_TFinfo_from_parquets() (in module `celloracle`.*motif_analysis*), 128

M

make_TFinfo_dataframe_and_dictionary() (in module `celloracle`.*motif_analysis.TFinfo method*), 126
make_TFinfo_from_scanned_file() (in module `celloracle`.*motif_analysis*), 128
makelog (class in `celloracle`.*utility*), 135
merged_score (in module `celloracle`.*Links attribute*), 115
merged_score (in module `celloracle`.*network_analysis.Links attribute*), 129
module
 celloracle, 115
 celloracle.data, 136
 celloracle.data_conversion, 136
 celloracle.go_analysis, 133
 celloracle.motif_analysis, 125
 celloracle.network_analysis, 129
 celloracle.utility, 134

N

name (in module `celloracle`.*Links attribute*), 115
name (in module `celloracle`.*network_analysis.Links attribute*), 129
Net (class in `celloracle`), 118

O

object_initiation_date (in module `celloracle`.*Net attribute*), 119
Oracle (class in `celloracle`), 121

P

palette (in module `celloracle`.*Links attribute*), 115
palette (in module `celloracle`.*network_analysis.Links attribute*), 129
peak2fasta() (in module `celloracle`.*motif_analysis*), 128
peak_df (in module `celloracle`.*motif_analysis.TFinfo attribute*), 125
plot_cartography_scatter_per_cluster() (in module `celloracle`.*Links method*), 116
plot_cartography_scatter_per_cluster() (in module `celloracle`.*network_analysis.Links method*), 130
plot_cartography_term() (in module `celloracle`.*Links method*), 116
plot_cartography_term() (in module `celloracle`.*network_analysis.Links method*), 130
plot_degree_distributions() (in module `celloracle`.*Links method*), 116

plot_degree_distributions() (*celloracle.network_analysis.Links method*), 131
 plot_mc_result_as_kde() (*celloracle.Oracle method*), 123
 plot_mc_result_as_trajectory() (*celloracle.Oracle method*), 123
 plot_mc_results_as_sankey() (*celloracle.Oracle method*), 124
 plot_network_entropy_distributions() (*celloracle.Links method*), 117
 plot_network_entropy_distributions() (*celloracle.network_analysis.Links method*), 131
 plot_score_comparison_2D() (*celloracle.Links method*), 117
 plot_score_comparison_2D() (*celloracle.network_analysis.Links method*), 131
 plot_score_distributions() (*celloracle.Links method*), 117
 plot_score_distributions() (*celloracle.network_analysis.Links method*), 131
 plot_score_per_cluster() (*celloracle.Links method*), 117
 plot_score_per_cluster() (*celloracle.network_analysis.Links method*), 131
 plot_scores_as_rank() (*celloracle.Links method*), 118
 plot_scores_as_rank() (*celloracle.network_analysis.Links method*), 132
 plotCoefs() (*celloracle.Net method*), 120
 prepare_markov_simulation() (*celloracle.Oracle method*), 124

R

read_bed() (*in module celloracle.motif_analysis*), 128
 ref_genome (*celloracle.motif_analysis.TFinfo attribute*), 126
 remove_zero_seq() (*in module celloracle.motif_analysis*), 128
 reset_dictionary_and_df() (*celloracle.motif_analysis.TFinfo method*), 126
 reset_filtering() (*celloracle.motif_analysis.TFinfo method*), 126
 run_markov_chain_simulation() (*celloracle.Oracle method*), 124

S

save_as_parquet() (*celloracle.motif_analysis.TFinfo method*), 126
 save_as_pickled_object() (*in module celloracle.utility*), 135
 scan() (*celloracle.motif_analysis.TFinfo method*), 126
 scan_dna_for_motifs() (*in module celloracle.motif_analysis*), 129

scanned_df (*celloracle.motif_analysis.TFinfo attribute*), 126
 set_R_path() (*in module celloracle.network_analysis*), 133
 seurat_object_to_anndata() (*in module celloracle.data_conversion*), 136
 simulate_shift() (*celloracle.Oracle method*), 124
 standard() (*in module celloracle.utility*), 135
 stats_dict (*celloracle.Net attribute*), 119
 summarize_mc_results_by_cluster() (*celloracle.Oracle method*), 125

T

test_R_libraries_installation() (*in module celloracle.network_analysis*), 133
 TFinfo (*celloracle.Net attribute*), 119
 TFinfo (*class in celloracle.motif_analysis*), 125
 to_dataframe() (*celloracle.motif_analysis.TFinfo method*), 127
 to_dictionary() (*celloracle.motif_analysis.TFinfo method*), 127
 to_hdf5() (*celloracle.Links method*), 118
 to_hdf5() (*celloracle.motif_analysis.TFinfo method*), 127
 to_hdf5() (*celloracle.Net method*), 121
 to_hdf5() (*celloracle.network_analysis.Links method*), 132
 to_hdf5() (*celloracle.Oracle method*), 125
 transfer_all_colors_between_anndata() (*in module celloracle.utility*), 135
 transfer_color_between_anndata() (*in module celloracle.utility*), 135
 transfer_scores_from_links_to_adata() (*in module celloracle.network_analysis*), 133

U

update_adata() (*in module celloracle.utility*), 136
 updateLinkList() (*celloracle.Net method*), 121
 updateTFinfo_dictionary() (*celloracle.Net method*), 121
 updateTFinfo_dictionary() (*celloracle.Oracle method*), 125