Supplementary file: AnanseScanpy equivalent

1. Python code AnanseScanpy

Python equivalent (in Jupyter Notebook) to the analysis steps in R. The vignette with full output is available on the GitHub page of AnanseScanpy.

1f. Install AnanseScanpy

```
mamba create -n anansescanpy anansescanpy
```

Part 3: Export single-cell cluster data

3a. Export cluster CPM, ATAC peak counts, and RNA-seq Counts

```
conda activate anansescanpy
pip install jupyter
jupyter notebook
```

From Jupyter notebook:

```
import scanpy as sc
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import anansescanpy as asc
```

```
# Fill in the directories where the h5ad RNA and ATAC objects are located
atac_PBMC=sc.read("atac_PBMC.h5ad")
rna_PBMC=sc.read("rna_PBMC.h5ad")

# Notes: the default Seurat assays for atac_PBMC and rna_PBMC are "peaks" and
"counts", respectively

# Necessary pre-processing from converted Seurat object
rna_PBMC.obs['predicted.id']=rna_PBMC.obs['predicted.id'].str.replace(' ', '-
')
atac_PBMC.obs['predicted.id']=atac_PBMC.obs['predicted.id'].str.replace(' ',
'-')
```

```
outputdir="AnanseScanpy_outs/"
contrasts=["B-naive_B-memory","B-memory_B-naive"]
minimal=25

asc.export_CPM_scANANSE(anndata=rna_PBMC,
min_cells=minimal,
outputdir=outputdir,
cluster_id="predicted.id"
)

asc.export_ATAC_scANANSE(anndata=atac_PBMC,
min_cells=minimal,
outputdir=outputdir,
cluster_id="predicted.id"
)

asc.config_scANANSE(anndata=rna_PBMC,
min_cells=minimal,
outputdir=outputdir,
cluster_id="predicted.id",
additional_contrasts=contrasts
)

asc.DEGS_scANANSE(anndata=rna_PBMC,
min_cells=minimal,
outputdir=outputdir,
cluster_id="predicted.id",
additional_contrasts=contrasts
)
```

Part 5: Import and visualise ANANSE results

5a. Import the ANANSE results

After running ANANSE with anansnake, the influence output is imported back into the single-cell object.

```
conda activate anansescanpy
jupyter notebook
```

From Jupyter notebook:

```
import scanpy as sc
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import anansescanpy as asc
```

```
# Extra step Python since h5ad objects were generated from diet Seurat objects
that do not have UMAP embeddings
# Generate a UMAP if not performed already during pre-processing

adata=rna_PBMC
adata.raw=adata
sc.pp.normalize_total(adata, target_sum=1e4)
sc.pp.log1p(adata)
sc.pp.pca(adata)
sc.pp.neighbors(adata, n_neighbors=10, n_pcs=30)
sc.tl.umap(adata)
```

```
# Import the ANANSE results to the Scanpy object and a separate data frame as
well
df_influence=asc.import_scanpy_scANANSE(
anndata=rna_PBMC,
cluster_id="predicted.id",
anansnake_inf_dir="AnanseScanpy_outs/influence/"
)
```

5b. Top five influential TFs per cluster

```
# Show the top five transcription factors for each population
top=5

df_t=df_influence.transpose()
factors_topn=[]
for i in df_t:
    df_sub=df_t[i]
    test=df_sub.sort_values(ascending=False)
    print(i,": ",list(test[0:top].index))
    factors_topn.append(list(test[0:top].index))

factors_topn=[j for i in factors_topn for j in i]
factors_topn=set(factors_topn)

selected_df=df_influence[list(factors_topn)]
```

5c. Heatmap most influential TFs

```
sns.clustermap(selected_df,
annot=False,
figsize=(15, 10)
)
```

5d. Visualise TF expression and influence on a UMAP

```
# Plot three TFs of interest upon the UMAP with expression and influence
scores
for i in ["STAT4","LEF1","MEF2C"]:
    fig, axs=plt.subplots(1,3, figsize=(20,5))
    sc.pl.umap(adata, color=[i], cmap="Blues",
        show=False,
        ax=axs[0])
    sc.pl.umap(adata, color=[str(i+"_influence")],
        cmap="Reds",
        show=False,
        ax=axs[1])
    sc.pl.umap(adata, color=["predicted.id"],
        show=False,
        ax=axs[2])
    plt.tight_layout()
```

Part 6: Specific cluster comparison

```
# Plot the direct cluster-to-cluster comparison
sc.set_figure_params(figsize=(10, 5))

MemoryInfluence=pd.read_csv('AnanseScanpy_outs/influence/anansesnake_B-
memory_B-naive.tsv',
sep="\t", header=0)
NaiveInfluence=pd.read_csv('AnanseScanpy_outs/influence/anansesnake_B-naive_B-
memory.tsv',
sep="\t", header=0)
NaiveInfluence["factor_fc"]=NaiveInfluence["factor_fc"]* -1
NaiveInfluence

B_comparison=pd.concat([NaiveInfluence, MemoryInfluence])
B_comparison=B_comparison.reset_index()
B_comparison

plt.style.use("ggplot")
plt.scatter(B_comparison["factor_fc"], B_comparison["influence_score"],
s=B_comparison["direct_targets"]*0.1, c=B_comparison["influence_score"],
cmap='Blues_r')

# Naming and adding range to x-axis
plt.xlabel('Expression Log2FC \n Naive B-cells / Memory B-cells')
plt.ylabel('Influence score')
plt.xlim([-2, 2])
plt.colorbar()
```

```
# Select factors with "factor_fc" > 0.26 and "factor_fc" < -0.5
selected_list=[i for i, x in enumerate(list(B_comparison["factor_fc"] > 0.26))
if x]+[i for i, x in enumerate(list(B_comparison["factor_fc"] < -0.5)) if x]

for i in selected_list:
    plt.annotate(B_comparison["factor"][int(i)],
    (B_comparison["factor_fc"][int(i)],
    B_comparison["influence_score"][int(i)])
    )

plt.figure()
```

Optional Part: Motif enrichment for predicting repressive factors

Import Motif enrichment scores

```
# Import the maelstrom results into the Scanpy object
asc.import_scanpy_maelstrom(anndata=adata,
cluster_id="predicted.id",
maelstrom_dir="AnanseScanpy_outs/maelstrom/"
)

# Make a dataframe with the values per cluster from the Scanpy object, like
df_maelstrom:
df_maelstrom=asc.per_cluster_df(anndata=adata,
assay="maelstrom",
cluster_id="predicted.id"
)

df_maelstrom.head()
```

Link TFs to motifs based on correlation coefficient

```
# Link motifs to transcription factors specified with "combine_motifs"
parameter
# Here, the maximum correlation of all motifs will be used (other options
include: max_var and means; see help)

adata=asc.Maelstrom_Motif2TF(anndata=adata,
cluster_id ="predicted.id",
maelstrom_dir="AnanseScanpy_outs/maelstrom/",
combine_motifs="max_cor",
save_output=True
)
```

Visualise TF expression and motif enrichment

Next the top TFs of either a negative or positive correlation can be visualised as a heatmap.

```
# Plotting the motif score heatmaps and the relative expression of top
negative correlating motifs
df_anticor=asc.per_cluster_df(anndata=adata,
assay="TFanticor_score",
cluster_id="predicted.id"
)

# Take a number of factors of interest or otherwise a top n factors:
factors_topn=["PAX5","STAT6","LMO2","SP1"]
factors_topn2=pd.Series(str(s) + '_TFanticor_score' for s in factors_topn)
selected_df=df_anticor[factors_topn2]
factors_topn2=factors_topn2.str.removesuffix('_TFanticor_score')
selected_df.columns=factors_topn2

df_expression=asc.per_cluster_df(anndata=adata,
assay="TFanticor_expression_score",
cluster_id="predicted.id"
)

factors_topn3=[str(s) + '_TFanticor_expression_score' for s in factors_topn]

selected_df2=df_expression[factors_topn3]

# Remove assay suffixes from Scanpy objects
factors_topn3=pd.Series(factors_topn3)
factors_topn3=factors_topn3.str.removesuffix('_TFanticor_expression_score')
selected_df2.columns=factors_topn3
```

```
# Plot the relative motif score map
res=sns.clustermap(selected_df, annot=False, figsize=(20, 10), cmap="PuOr_r")

# Reorder heatmaps according to the other one above
selected_df2=selected_df2[list(selected_df.columns[res.dendrogram_col.reordere
d_ind])]
selected_df2=selected_df2.reindex(list(selected_df.index[res.dendrogram_row.re
ordered_ind]))

# Plot the relative expression score map
sns.clustermap(selected_df2, annot=False, figsize=(20, 10), col_cluster=False,
row_cluster=False, cmap="RdBu_r")
```

```
# Plotting the motif score heatmaps and the relative expression of top
positive correlating motifs
df_cor=asc.per_cluster_df(anndata=adata,
assay="TFcor_score",
cluster_id="predicted.id"
)

# Take a number of factors of interest or otherwise a top n factors:
factors_topn=["MEF2C","ETS1","RXRA","FOSL2"]
factors_topn2=pd.Series(str(s) + '_TFcor_score' for s in factors_topn)
selected_df=df_cor[factors_topn2]
factors_topn2=factors_topn2.str.removesuffix('_TFcor_score')
selected_df.columns=factors_topn2

df_expression=asc.per_cluster_df(anndata=adata,
assay="TFcor_expression_score",
cluster_id="predicted.id"
)

factors_topn3=[str(s) + '_TFcor_expression_score' for s in factors_topn]

selected_df2=df_expression[factors_topn3]

# Remove assay suffixes from Scanpy objects
factors_topn3=pd.Series(factors_topn3)
factors_topn3=factors_topn3.str.removesuffix('_TFcor_expression_score')
selected_df2.columns=factors_topn3

# Plot the relative motif score map
res=sns.clustermap(selected_df, annot=False, figsize=(20, 10),cmap="PuOr_r")
```

```
# Reorder heatmaps according to the other one above
selected_df2=selected_df2[list(selected_df.columns[res.dendrogram_col.reordere
d_ind])]
selected_df2=selected_df2.reindex(list(selected_df.index[res.dendrogram_row.re
ordered_ind]))

# Plot the relative expression score map
sns.clustermap(selected_df2, annot=False, figsize=(20, 10),col_cluster=False,
row_cluster=False, cmap="RdBu_r")
```

```
# Showing the negative correlating factors of interest with "max_cor" as the
default "combine_motifs" parameter

TF_list=["STAT6","PAX5"]
asc.Factor_Motif_Plot(adata,
factor_list=TF_list,
logo_dir="AnanseScanpy_outs/maelstrom/logos/",
assay_maelstrom ="TFanticor"
)
```