Please note that codes and reports of Q1 and Q3 were written together in Jupyter Notebooks. It is advised to view the reports for these two questions in the notebooks rather than this PDF. (except for Q3 part 1 which is on page 19)

# bio-hw4-q1-new

December 22, 2017

# 1 Intro to Bioinformatics

## 1.1 Homework Assignment 4

## 1.2 Question 1: Feedback, Robustness & Bifurcation

## 1.3 Sepehr Torabparhiz

## 1.4 93100774

### 1.4.1 Developed by Python 3.6

```
In [1]: import numpy as np
        from matplotlib import pyplot as plt
        import seaborn as sb

        import scipy
        import scipy.optimize

        %matplotlib inline
        %config InlineBackend.figure_format = 'retina'

        import sympy as sp
        sp.init_printing()
```

## 1.5 Finding the initial densities & defining the equations

I use *SymPy* to represent the density change rates in Python. First, I define the densities by using *symbols*.

```
In [2]: X, Y, Z = sp.symbols('X Y Z')
```

One by one, I find the initial densities. As the system is said to be stable at the start of the experiment, the density change rates should be zero. Also, alpha is 1.6 at the start.

```
In [3]: X0 = sp.solve(1 - 1.6 * X)
        x0 = X0[0]
        x0
```

Out[3]:

$$0.625$$

Y and Z equations have multiple real and complex solutions. I choose the real solutions as the initial densities.

```
In [4]: Y0 = sp.solve(0.5 + 1 / (Y**4 + 1) - 1.6 * Y)
        y0 = Y0[0]
        Y0
```

Out[4]:

$$[0.773037438345376, \quad -0.805422290186745 - 0.735072417370233i, \quad -0.805422290186745 + 0.735072417370233i,$$

As it can be seen above, there are multiple solutions for the equation. I only use the first one which is real. If there were two solutions that would be a case of bifurcation.

```
In [5]: Z0 = sp.solve(0.5 + ( Z**4 / (Z**4 + 1)) - 1.6 * Z)
        z0 = Z0[0]
        z0
```

Out[5]:

$$0.318897592409502$$

The protein density at each minute is computed by finding the density change in the previous minute and adding that change to the previous density. Below, I define the functions that compute the density change.

```
In [6]: dx = lambda x, alpha: (1 - alpha * x)
        dy = lambda y, alpha: (0.5 + 1 / (y**4 + 1) - alpha * y)
        dz = lambda z, alpha: (0.5 + (z**4) / (z**4 + 1) - alpha * z)
```

The value of alpha in every minute of the 48 hours is generated.

```
In [7]: alpha_48 = np.tile(np.concatenate((np.arange(1.6, 0.4, -0.1 / 60)[:-1], np.arange(0.4, 1
```

```
In [8]: x_densities = [x0]
        y_densities = [y0]
        z_densities = [z0]
```

Now, for every minute the density of each protein is calculated. New values are found by adding the density change to the previous density.

But, this method produced bad results for Y densities, so I used SciPy's *fsolve* to actually setting the left side of Y's equation to zero and solving the equation each time.

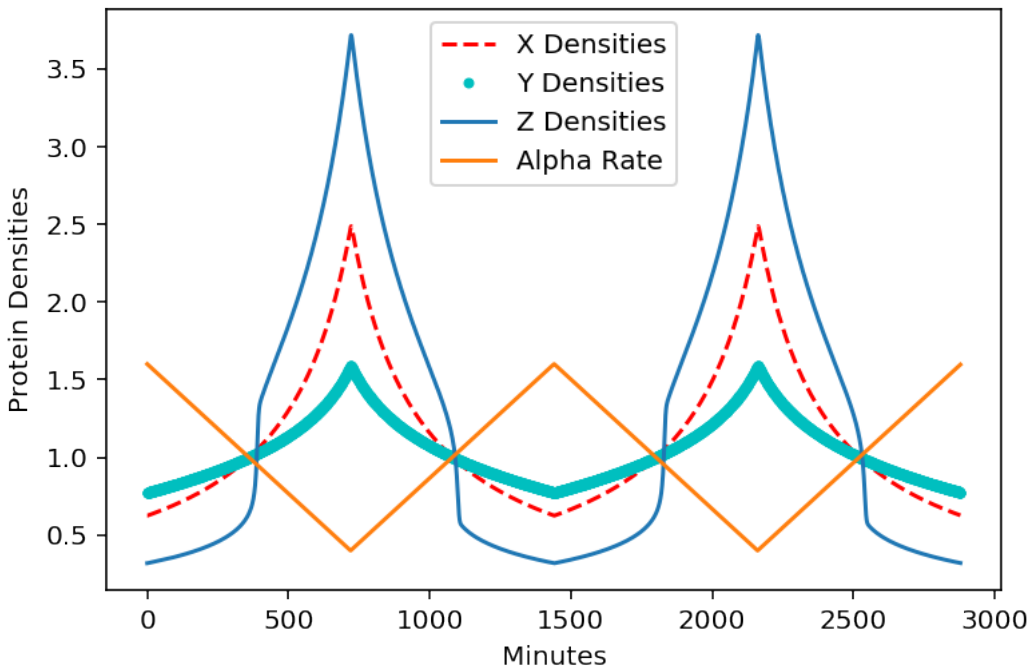
Also if the density falls below zero in a time step, it is set to zero in the density lists.

```
In [9]: %%time
        i = 1
        for alpha in alpha_48[:2880]:
            prev_x_density = x_densities[i-1]
            prev_y_density = y_densities[i-1]
            prev_z_density = z_densities[i-1]

            new_x_density = prev_x_density + dx(prev_x_density, alpha)

            new_y_density = scipy.optimize.fsolve(
                lambda y: 0.5 + 1 / (y**4 + 1) - alpha * y, float(prev_y_density))[0]

            new_z_density = prev_z_density + dz(prev_z_density, alpha)

            x_densities.append(new_x_density)
            y_densities.append(new_y_density)
            z_densities.append(new_z_density)

            i += 1
```

```
CPU times: user 2.48 s, sys: 28.5 ms, total: 2.51 s
Wall time: 2.5 s
```

```
In [10]: # The x values for the plots
         minutes = list(range(60*48))
```

```
In [11]: plt.plot(minutes, x_densities[:2880], 'r--', label='X Densities')
         plt.plot(minutes, y_densities[:2880], 'c.', label='Y Densities')
         plt.plot(minutes, z_densities[:2880], label='Z Densities')
         plt.plot(minutes, alpha_48[:2880], label='Alpha Rate')
         plt.xlabel('Minutes')
         plt.ylabel('Protein Densities')
         plt.legend(loc='upper center')
         plt.show()
```

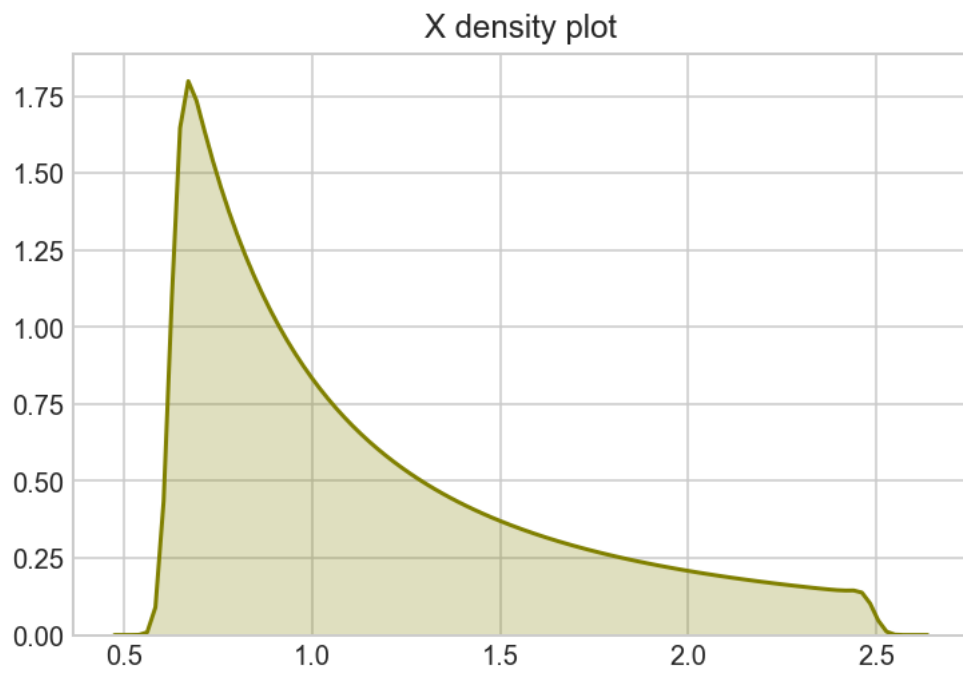



Each protein's density increases as alpha decreases and vice verca.
Z has the highest variance in density while Y changes the least.

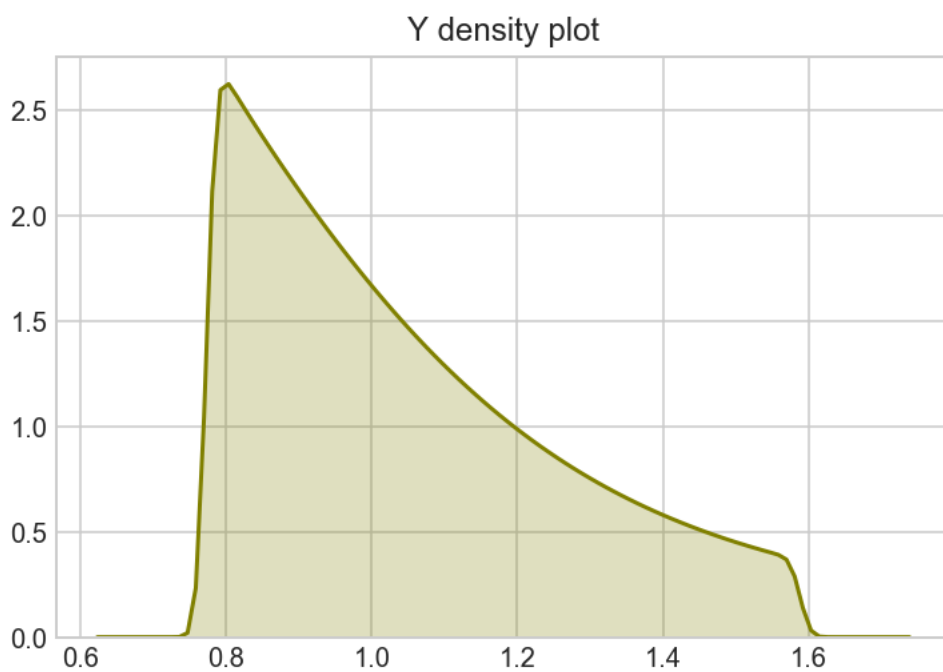
### 1.5.1 Drawing the density plots

In the previous step, the densities of each protein through the 48 hours are stored in different lists. Using these lists and *Seaborn*'s *kdeplot*, the density plot for each protein is drawn.

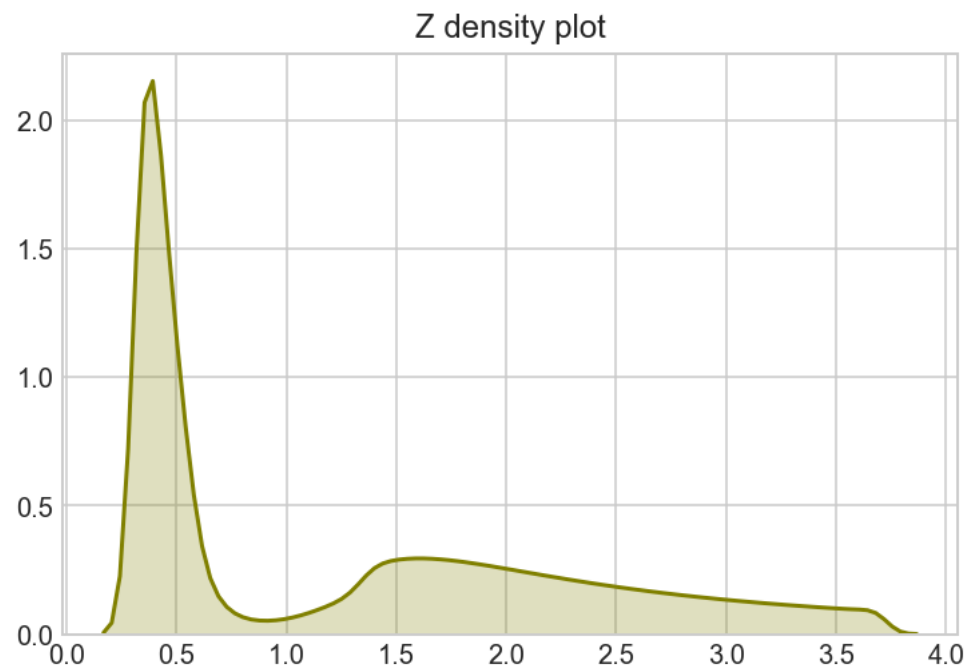
```
In [12]: sb.set_style('whitegrid')
         sb.kdeplot(np.array(x_densities), shade=True, bw=.05, color="olive").set_title('X densi

Out[12]: Text(0.5,1,'X density plot')
```




```
In [13]: sb.set_style('whitegrid')
         sb.kdeplot(np.array(y_densities), shade=True, bw=.05, color="olive").set_title('Y densi

Out[13]: Text(0.5,1,'Y density plot')
```



```
In [14]: sb.set_style('whitegrid')
         sb.kdeplot(np.array(z_densities), shade=True, bw=.05, color="olive").set_title('Z densi

Out[14]: Text(0.5,1,'Z density plot')
```




Density plots of X and Y are quite similar. X's plot has a steeper slope in the beginning which decrease in higher densities, while Y's plot has a more consistent slope. We can argue that Y is more robust than X and Z.

Compared to other proteins, Z's densities are concentrated in a small range(between 0.2 and 0.7). Z also has the steepest slope among these 3 proteins.

Sepehr Torabparhiz

93100774

December 22, 2017

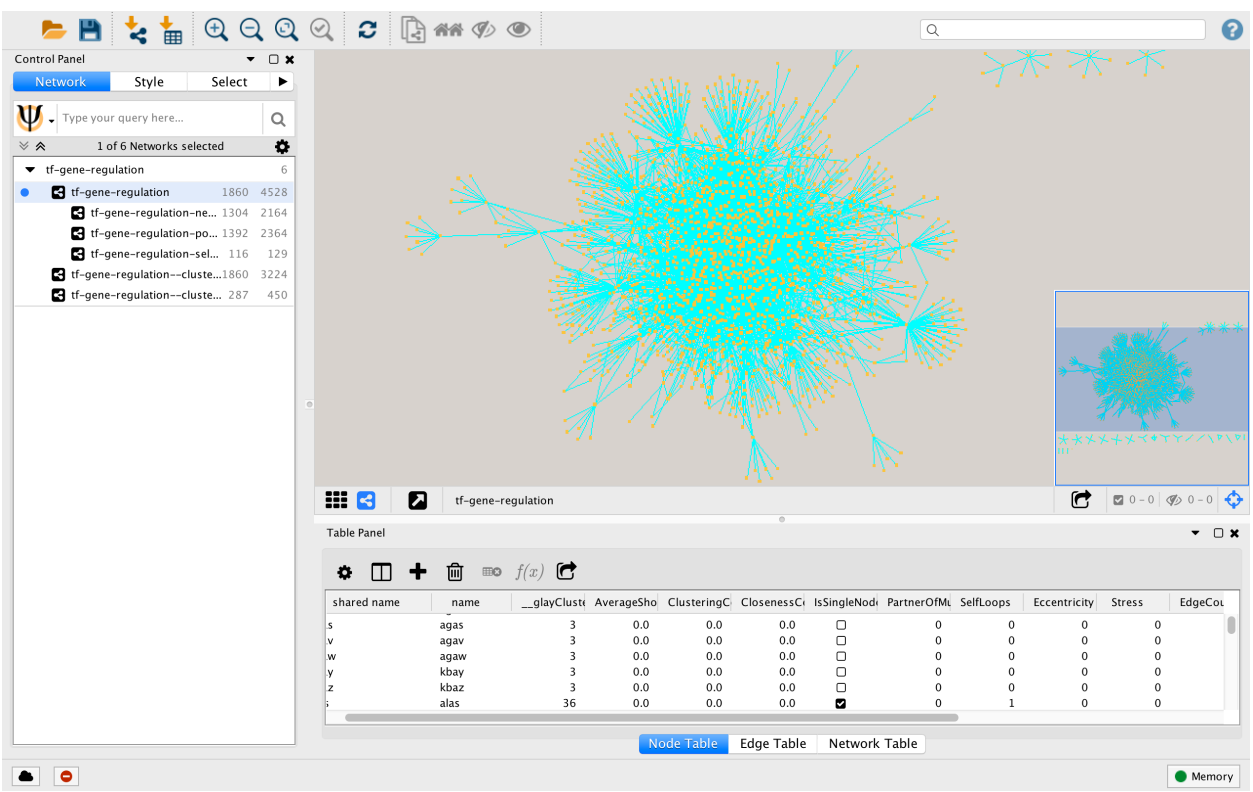
# Intro to Bioinformatics

## Homework Assignment 4 - Question 2

After downloading the TF-Gene file from RegulonDB, I read the text file with python and split each line to different entries. Then, I saved the list of split lines in a Pandas dataframe.

Next, I cleaned up the data by setting names for each column, setting all gene and TF names to lowercase and removing entries whose Evidence-Type was null or Regulatory-Effect was ?.

At last, for each row whose Regulatory-Effect was +-, I duplicated that row and changed the Regulatory-Effects so for each such row there is two rows. One with + and the other with - for Regulatory-Effect. At last I saved the dataframe in an Excel file. Screen shots of the code is attached to the end of this document.



Then, I imported the clean data into Cytoscape, setting the Regulatory-Effect as the interaction type of each edge.

A) Using the NetworkAnalyzer, I found the following facts about the network

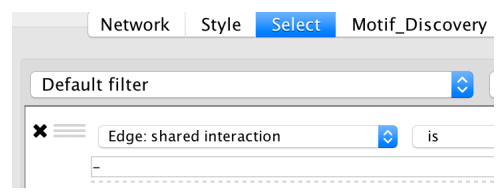
Number of Nodes: 1860

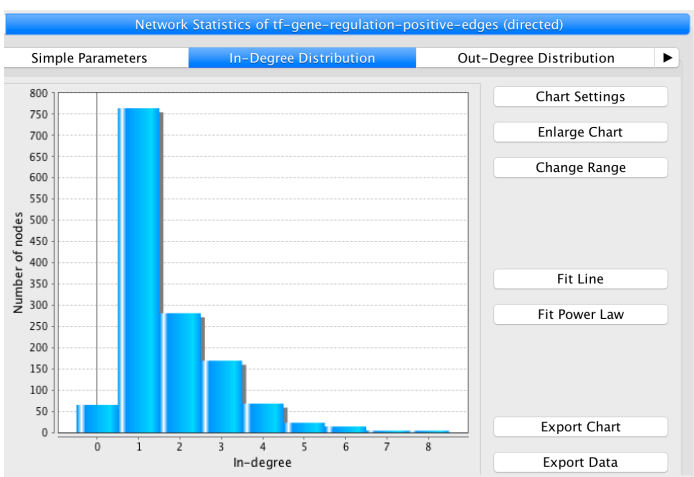
Number of Edges: 4528

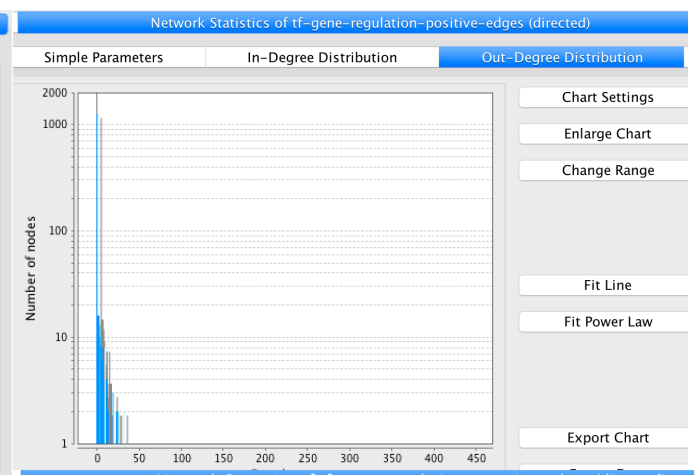
| | | | |
|---|---|---|---|
| Clustering coefficient : | **0.113** | Number of nodes : | **1860** |
| Connected components : | **25** | Network density : | **0.0** |
| Network diameter : | **10** | Isolated nodes : | **1** |
| Network radius : | **1** | Number of self-loops : | **129** |
| Shortest paths : | **19443 (0%)** | Multi-edge node pairs : | **208** |
| Characteristic path length : | **2.957** | Analysis time (sec) : | **2.705** |
| Avg. number of neighbors : | **4.452** | | |

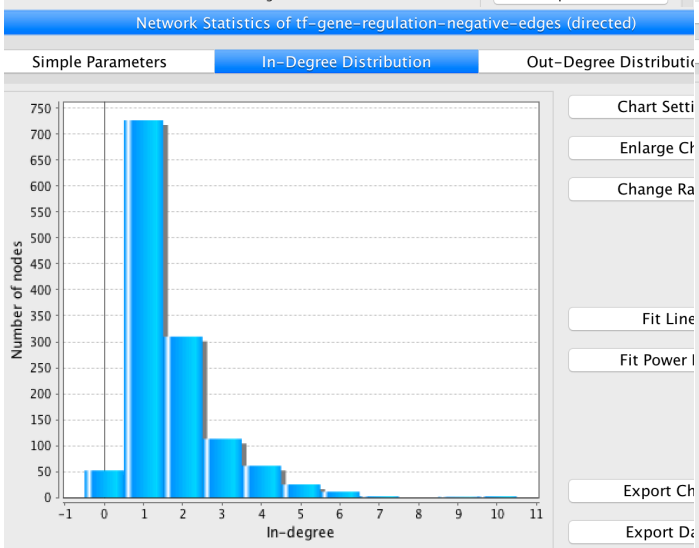
**In-Degree & Out-Degree distributions for positive and negative edges:**

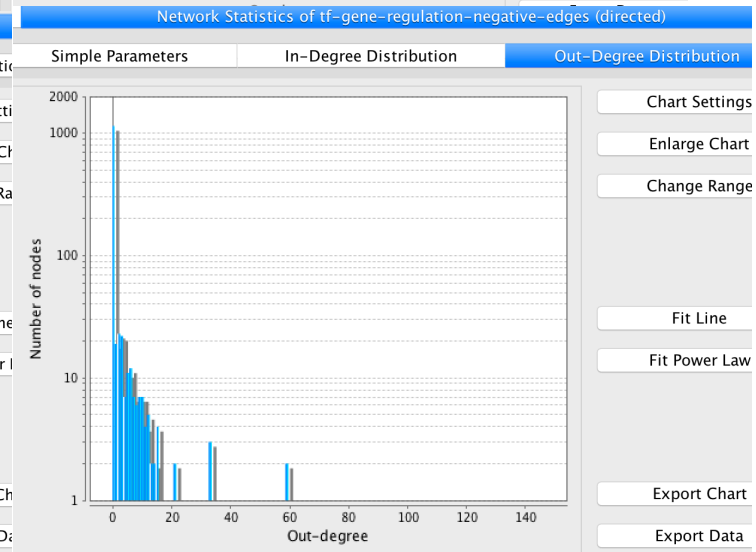
I used Cytoscape's filtering ability to select negative and positive edges and create different networks for each edge type. Then, I ran the NetworkAnalyzer on each network to find the distributions. (Out-Degree Y axis is logarithmic)



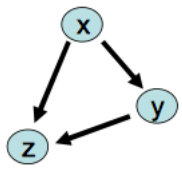



B)

Negative Auto Regulatory are Nodes that are self-looping and their loop's edge type is -

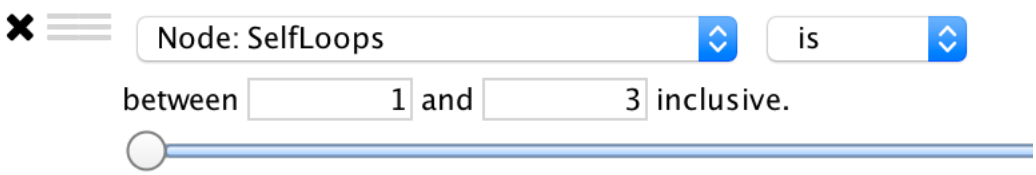
Positive Auto Regulatory are Nodes that are self-looping and their loop's edge type is +

Feed Forward Loops are subgraphs which look like the picture below.

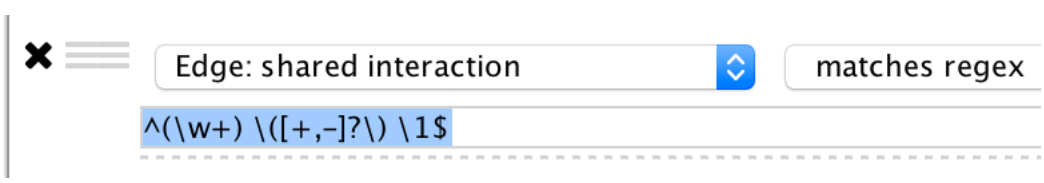



Feed-forward loop
(FFL)

First, I selected self-looping nodes with the filer below made a new network from them.




Then I deleted the non-looping edges from this network. I selected all edges whose shared interaction name was a string that contained the same gene name with a (+) or a (-) between them. I achieved this by using a regex pattern.

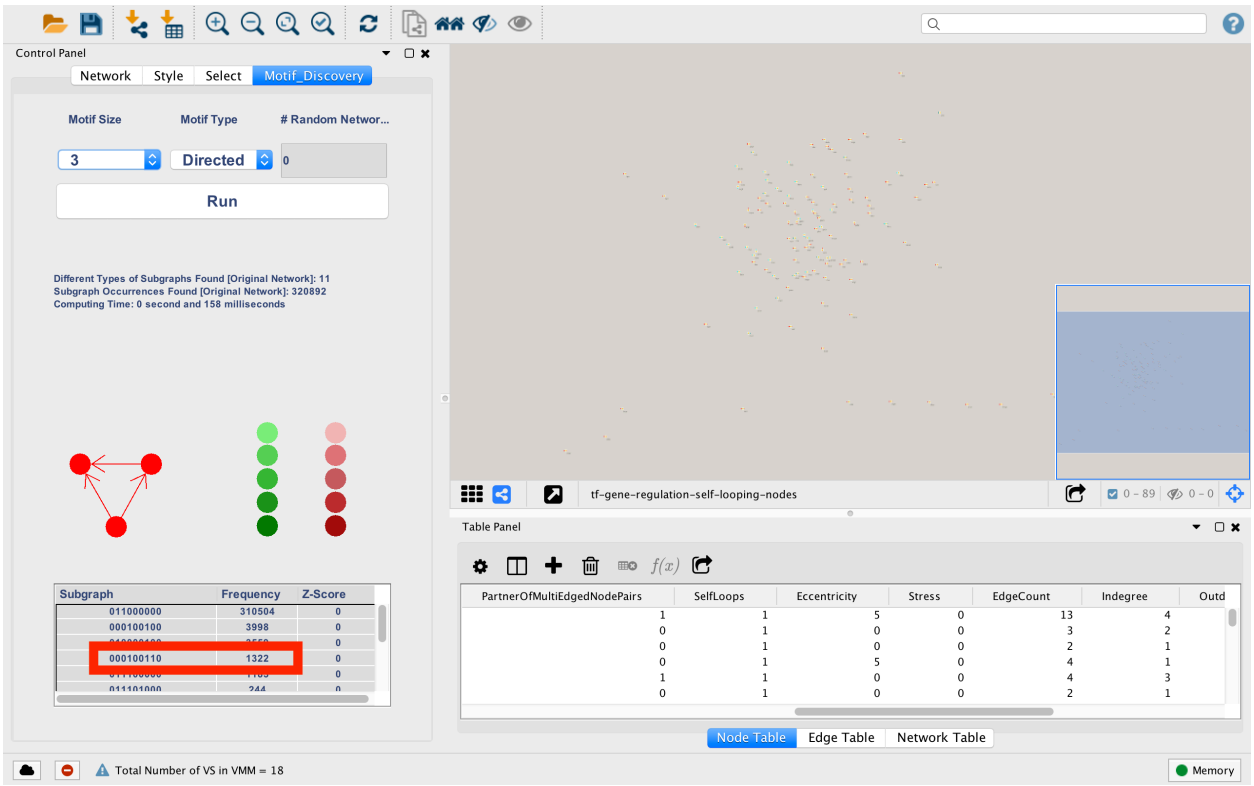



And now I had a network of self-looping nodes and their looping edges. Again I used the filters to find out how many of these looping edges were + or -.

PAR or + looping edge count: 40

NAR or - looping edge count: 89

In order to find the FFL subgraphs I installed the Motif-Discovery app on Cytoscape. I set the motif size to 3 and the motif type to Directed. after running the app I chose the motif which was of the feed forward loop type and clicked on it so it was drawn.



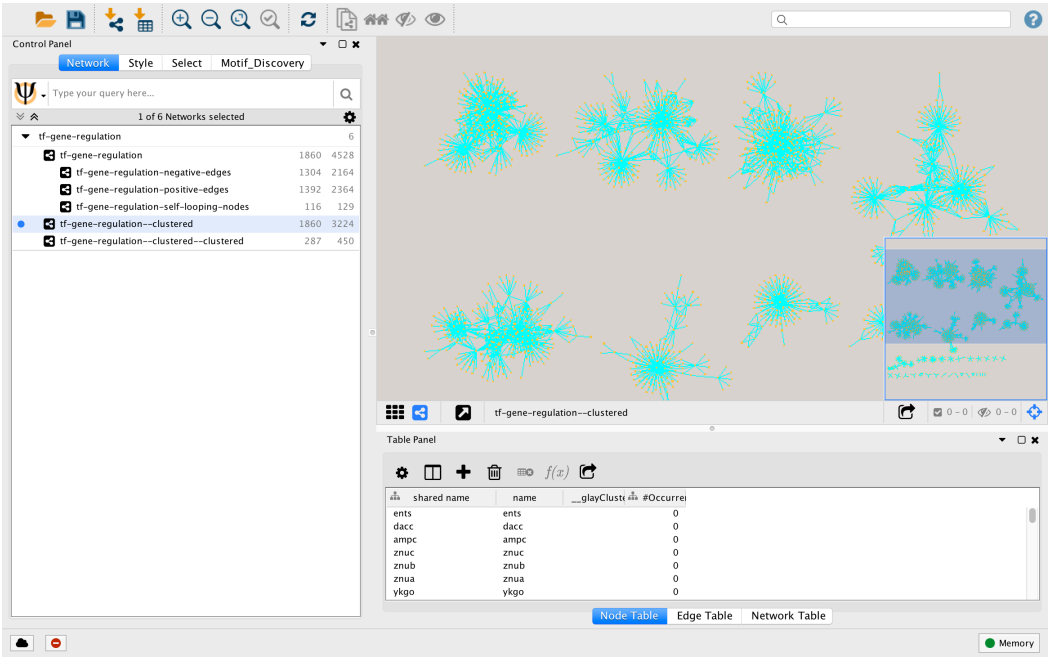

As it can be seen above, there were 1322 FFLs in the network.

C)

Number of Nodes in the second cluster: 287

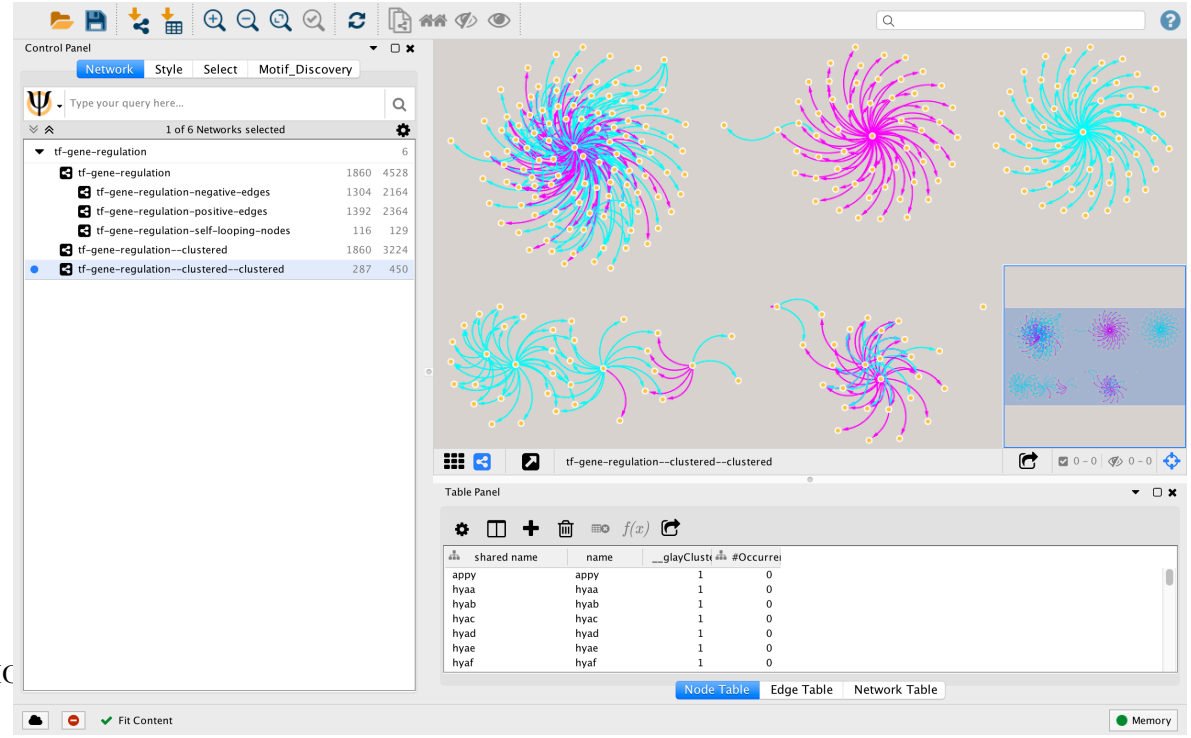
Number of Edges in the second cluster: 450

First Clustering:



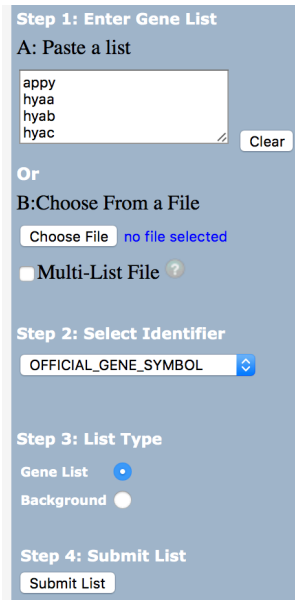

Second Clustering:

Neon Blue edges are + and purple edges are -.

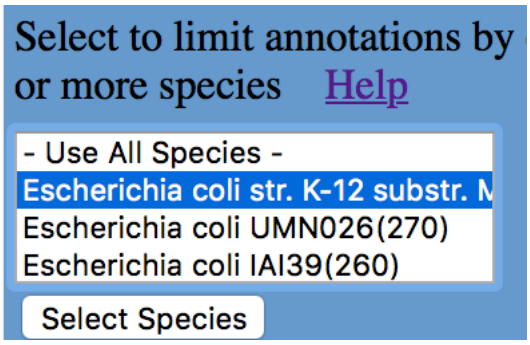


D)

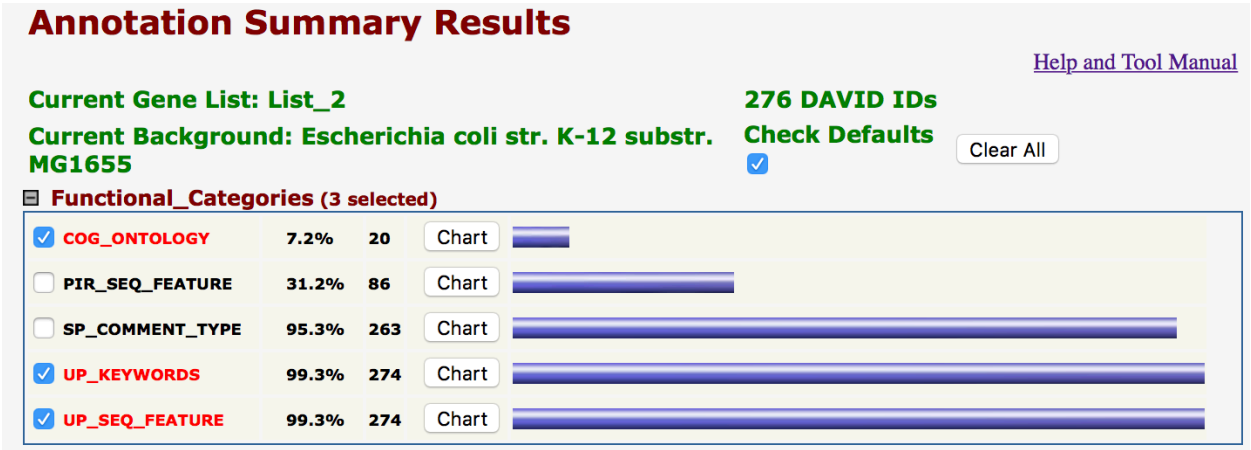
First, I exported the table of the second cluster to a CSV file. Then, I opened this file in Excel and copied all the gene names in it and pasted these names in DAVID. I chose the OFFICIAL-GENE_SYMBOL setting and submitted the list.




Then, DAVID showed a list of possible species from which I chose the first one.




From the Functional Categories list, DAVID drew charts for UP_KEYWORDS and UP_SEQ_FEATURE. I sorted the charts by the number of genes to see what functional annotation most of the genes had.



| Category | Term | RT | Genes | Count | % | P-Value | Benjamini |
|---|---|---|---|---|---|---|---|
| UP_KEYWORDS | Membrane | RT | | 92 | 33.3 | 3.6E-2 | 1.8E-1 |
| UP_KEYWORDS | Cell membrane | RT | | 88 | 31.9 | 7.8E-3 | 4.7E-2 |
| UP_KEYWORDS | Metal-binding | RT | | 84 | 30.4 | 3.1E-11 | 5.0E-10 |
| UP_KEYWORDS | Cell inner membrane | RT | | 80 | 29.0 | 1.8E-3 | 1.2E-2 |
| UP_KEYWORDS | Transmembrane | RT | | 73 | 26.4 | 7.4E-2 | 3.1E-1 |
| UP_KEYWORDS | Transmembrane helix | RT | | 71 | 25.7 | 5.4E-2 | 2.5E-1 |
| UP_KEYWORDS | Transport | RT | | 69 | 25.0 | 8.4E-4 | 5.8E-3 |
| UP_KEYWORDS | Oxidoreductase | RT | | 62 | 22.5 | 2.8E-13 | 6.7E-12 |
| UP_KEYWORDS | Iron | RT | | 61 | 22.1 | 2.9E-22 | 2.1E-20 |

| Category | Term | RT | Genes | Count | % | P-Value | Benjamini |
|---|---|---|---|---|---|---|---|
| UP_SEQ_FEATURE | transmembrane region | RT | | 70 | 25.4 | 4.9E-2 | 7.8E-1 |
| UP_SEQ_FEATURE | topological domain:Cytoplasmic | RT | | 47 | 17.0 | 8.5E-2 | 9.1E-1 |
| UP_SEQ_FEATURE | mutagenesis site | RT | | 37 | 13.4 | 8.8E-3 | 2.8E-1 |
| UP_SEQ_FEATURE | metal ion-binding site:Iron-sulfur 2 (4Fe-4S) | RT | | 19 | 6.9 | 5.5E-12 | 2.5E-9 |
| UP_SEQ_FEATURE | metal ion-binding site:Iron-sulfur 1 (4Fe-4S) | RT | | 17 | 6.2 | 1.8E-10 | 4.1E-8 |
| UP_SEQ_FEATURE | domain:4Fe-4S ferredoxin-type 1 | RT | | 16 | 5.8 | 4.6E-10 | 6.9E-8 |
| UP_SEQ_FEATURE | domain:4Fe-4S ferredoxin-type 2 | RT | | 16 | 5.8 | 4.6E-10 | 6.9E-8 |

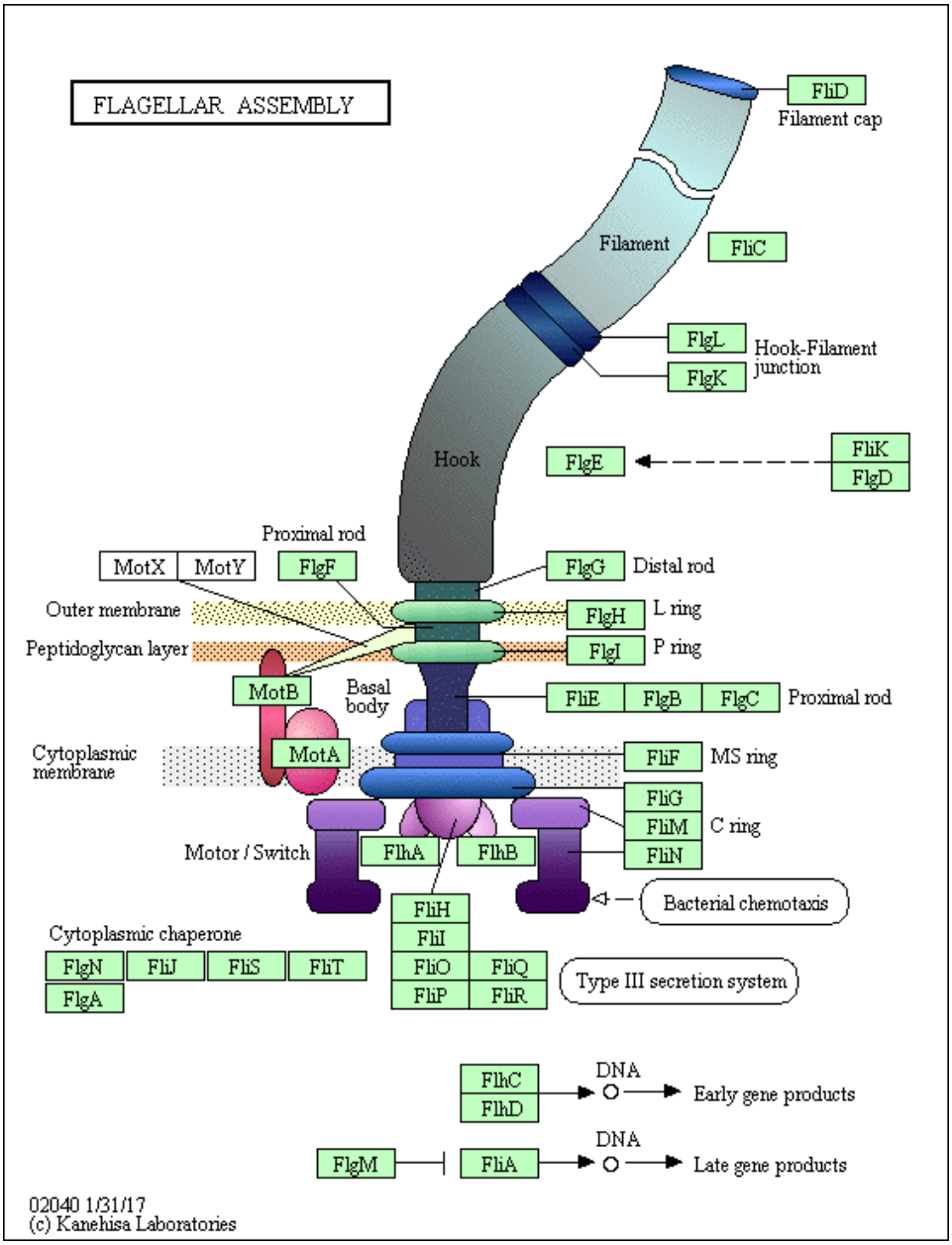
It can be seen that these genes affect the cell membrane, iron-binding proteins and in general how the cell transports metals like iron through the membrane.

As for the pathway lists, DAVID shows that these genes affect the two component system and the flagellar assembly which forms the flagellum organelle which helps E.coli bacteria to move around.

| Category | Term | RT | Genes | Count |
|---|---|---|---|---|
| KEGG_PATHWAY | Two-component system | RT | | 34 |
| KEGG_PATHWAY | Flagellar assembly | RT | | 22 |

**Pathways (1 selected)**

| | | | | |
|---|---|---|---|---|
| EC_NUMBER | 37.7% | 104 | Chart | |
| KEGG_PATHWAY | 48.2% | 133 | Chart | |


FLAGELLAR ASSEMBLY
FliD
Filament cap
Filament
FliC
FlgL
FlgK
Hook-Filament junction
Hook
FlgE
FliK
FlgD
Proximal rod
MotX
MotY
FlgF
FlgG
Distal rod
Outer membrane
FlgH
L ring
Peptidoglycan layer
FlgI
P ring
MotB
Basal body
FliE
FlgB
FlgC
Proximal rod
Cytoplasmic membrane
MotA
FliF
MS ring
FliG
FliM
FliN
C ring
Motor / Switch
FlhA
FlhB
Bacterial chemotaxis
FliH
FliI
FliO
FliP
FliQ
FliR
Type III secretion system
Cytoplasmic chaperone
FlgN
FliJ
FliS
FliT
FlgA
FlhC
FlhD
DNA
Early gene products
FlgM
FliA
DNA
Late gene products
02040 1/31/17
(c) Kanehisa Laboratories

# bio-hw4-q2-GRN

December 22, 2017

```
In [1]: import pandas as pd
        import numpy as np

In [2]: tf_gene_pd = pd.read_pickle('tf-gene-dataframe.pickle')

In [3]: tf_gene_pd.head(5)

Out[3]:   TF-Name Regulated-Gene Regulatory-Effect                        Evidence  \
        0    AccB           accB                 -                               []
        1    AccB           accC                 -                               []
        2    AcrR           acrA                 -          [BCE, BPP, GEA, HIBSCS]
        3    AcrR           acrB                 -          [BCE, BPP, GEA, HIBSCS]
        4    AcrR           acrR                 -  [AIBSCS, BCE, BPP, GEA, HIBSCS]

          Evidence-Type
        0          null
        1          null
        2          Weak
        3          Weak
        4          Weak

In [4]: ev_col = tf_gene_pd['Evidence']

In [5]: new_col = [r.replace('[', '').replace(']', '') for r in ev_col]

In [6]: new_col[:5]

Out[6]: ['',
         '',
         'BCE, BPP, GEA, HIBSCS',
         'BCE, BPP, GEA, HIBSCS',
         'AIBSCS, BCE, BPP, GEA, HIBSCS']

In [7]: tf_gene_pd['Evidence'] = new_col

In [8]: tf_gene_pd.head(10)
```

```
Out[8]:   TF-Name Regulated-Gene Regulatory-Effect                      Evidence  \
        0    AccB           accB                 -
        1    AccB           accC                 -
        2    AcrR           acrA                 -          BCE, BPP, GEA, HIBSCS
        3    AcrR           acrB                 -          BCE, BPP, GEA, HIBSCS
        4    AcrR           acrR                 -  AIBSCS, BCE, BPP, GEA, HIBSCS
        5    AcrR           flhC                 -                     GEA, HIBSCS
        6    AcrR           flhD                 -                     GEA, HIBSCS
        7    AcrR           marA                 -               BPP, GEA, HIBSCS
        8    AcrR           marB                 -               BPP, GEA, HIBSCS
        9    AcrR           marR                 -               BPP, GEA, HIBSCS

          Evidence-Type
        0          null
        1          null
        2          Weak
        3          Weak
        4          Weak
        5          Weak
        6          Weak
        7        Strong
        8        Strong
        9        Strong
```

```
In [9]: tf_gene_pd.to_excel('tf-gene-cleaned.xlsx', sheet_name='tf-gene-regulation', index=False
```

```
In [10]: tf_gene_pd['TF-Name'] = tf_gene_pd['TF-Name'].str.lower()
         tf_gene_pd['Regulated-Gene'] = tf_gene_pd['Regulated-Gene'].str.lower()
```

```
In [11]: tf_gene_pd.to_excel('tf-gene-lowered-cleaned.xlsx', sheet_name='tf-gene-regulation', in
```

```
In [12]: np.unique(tf_gene_pd['Evidence-Type'])
```

```
Out[12]: array(['Strong', 'Weak', 'null'], dtype=object)
```

```
In [13]: np.unique(tf_gene_pd['Regulatory-Effect'])
```

```
Out[13]: array(['+', '+-', '-', '?'], dtype=object)
```

```
In [14]: # Remove ?s and nulls
         na_less = tf_gene_pd[tf_gene_pd['Regulatory-Effect'] != '?']
         na_less = na_less[na_less['Evidence-Type'] != 'null']
```

```
In [15]: na_less.head(10)
```

```
Out[15]:    TF-Name Regulated-Gene Regulatory-Effect                      Evidence  \
         2     acrr           acra                 -          BCE, BPP, GEA, HIBSCS
         3     acrr           acrb                 -          BCE, BPP, GEA, HIBSCS
         4     acrr           acrr                 -  AIBSCS, BCE, BPP, GEA, HIBSCS
```

```
5     acrr          flhc                -                    GEA, HIBSCS
6     acrr          flhd                -                    GEA, HIBSCS
7     acrr          mara                -               BPP, GEA, HIBSCS
8     acrr          marb                -               BPP, GEA, HIBSCS
9     acrr          marr                -               BPP, GEA, HIBSCS
10    acrr          micf                -                         AIBSCS
11    acrr          soxr                -               BPP, GEA, HIBSCS

   Evidence-Type
2           Weak
3           Weak
4           Weak
5           Weak
6           Weak
7         Strong
8         Strong
9         Strong
10          Weak
11        Strong
```

In [16]:
```
na_less.to_excel('tf-gene-nullCleaned-lowered-cleaned.xlsx', sheet_name='tf-gene-regula
```

In [17]:
```
duals = na_less[na_less['Regulatory-Effect'] == '+-']
```

In [18]:
```
len(na_less)
```

Out[18]:
```
4457
```

In [19]:
```
cnt = 0
lind = duals.index[-1]
dd = pd.DataFrame(columns=list(duals.columns))
news = []
for ind, row in duals.iterrows():
    nr = pd.DataFrame(
            [[row['TF-Name'], row['Regulated-Gene'], '-', row['Evidence'], row['Evidenc
    dd = dd.append(nr)
    row['Regulatory-Effect'] = '+'
    cnt += 1
    lind += 1

dual_fixed = pd.concat([duals, dd])
```

In [20]:
```
df_full = pd.concat([na_less[na_less['Regulatory-Effect'] != '+-'], dual_fixed])
```

In [25]:
```
df_full.to_excel('tf-gene-final.xlsx', sheet_name='tf-gene-regulation', index=False)
```

In [21]:
```
np.unique(df_full['Regulatory-Effect'])
```

Out[21]:
```
array(['+', '-'], dtype=object)
```

In [24]: df_full.head(10)

Out[24]:

| | TF-Name | Regulated-Gene | Regulatory-Effect | Evidence | Evidence-Type |
|---|---|---|---|---|---|
| 2 | acrr | acra | - | BCE, BPP, GEA, HIBSCS | Weak |
| 3 | acrr | acrb | - | BCE, BPP, GEA, HIBSCS | Weak |
| 4 | acrr | acrr | - | AIBSCS, BCE, BPP, GEA, HIBSCS | Weak |
| 5 | acrr | flhc | - | GEA, HIBSCS | Weak |
| 6 | acrr | flhd | - | GEA, HIBSCS | Weak |
| 7 | acrr | mara | - | BPP, GEA, HIBSCS | Strong |
| 8 | acrr | marb | - | BPP, GEA, HIBSCS | Strong |
| 9 | acrr | marr | - | BPP, GEA, HIBSCS | Strong |
| 10 | acrr | micf | - | AIBSCS | Weak |
| 11 | acrr | soxr | - | BPP, GEA, HIBSCS | Strong |

In [22]: pd.DataFrame(
[[row['TF-Name'], row['Regulated-Gene'], '-', row['Evidence'], row['Evidenc

Out[22]:

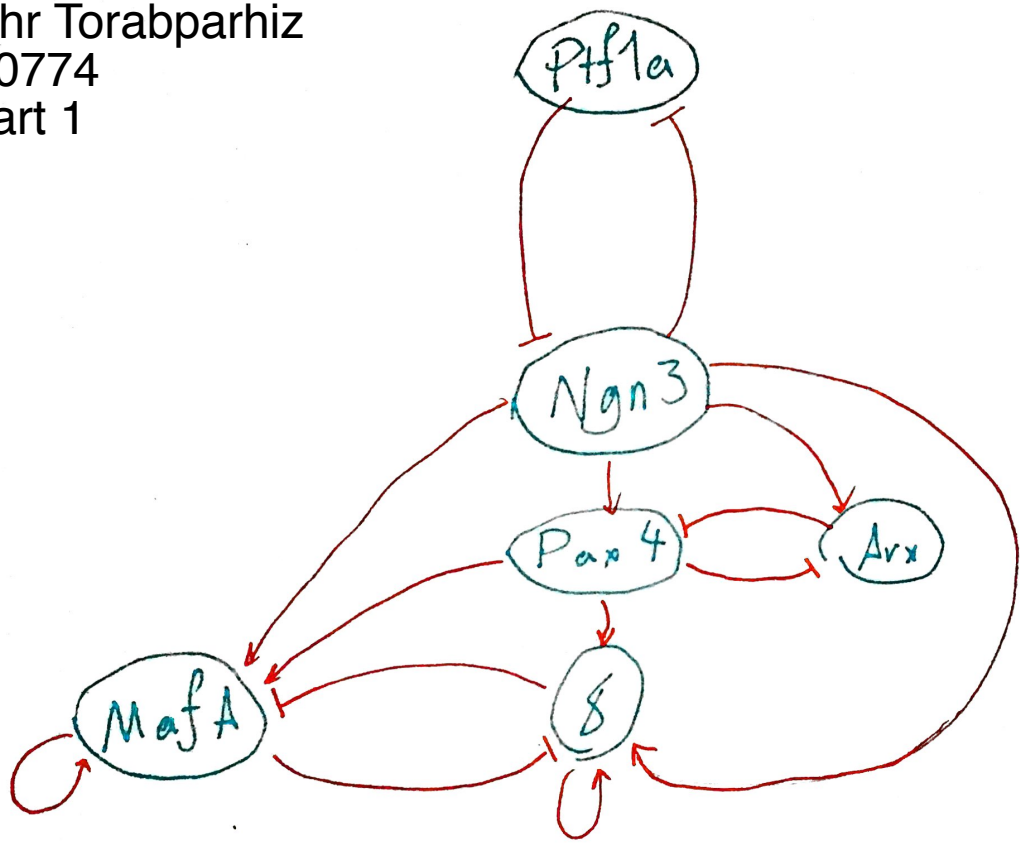
| | TF-Name | Regulated-Gene | Regulatory-Effect | Evidence | Evidence-Type |
|---|---|---|---|---|---|
| 0 | ompr | ompf | - | AIBSCS, BCE, BPP, GEA, HIBSCS, SM | Weak |

In [23]: nr

Out[23]:

| | TF-Name | Regulated-Gene | Regulatory-Effect | Evidence | Evidence-Type |
|---|---|---|---|---|---|
| 0 | ompr | ompf | - | AIBSCS, BCE, BPP, GEA, HIBSCS, SM | Weak |

Sepehr Torabparhiz
93100774
Q3 part 1




If Ngn3 wins Pax4/Arx and MafA/Delta switches have a chance to function, so Ngn3 is an activator for all of them. Same is true for Pax4 activating MafA and Delta.

For each switch, the gene pair inhibit one another as they compete.

And for MafA & delta, we can argue that they activate themselves to keep make the final cell type which the Beta or the Delta cell.

# bio-hw4-q3

December 23, 2017

# 1 Intro to Bioinformatics

## 1.1 Homework Assignment 4

## 1.2 Question 3: Hierarchical Attractors

## 1.3 Sepehr Torabparhiz

## 1.4 93100774

### 1.4.1 Developed by Python 3.6

```
In [1]: import numpy as np
```

### 1.4.2 Computing Probabilites for each switch

```
In [2]: endocrine_progenitor_prob = lambda ngn3, ptf1a: 0.001 * (
            (ngn3**2 + ngn3) / (ngn3**2 + 1)) * (
            (ptf1a**2 + 1) / (ptf1a**2 + ptf1a + 1))

        beta_delta_progenitor_prob = lambda pax4, arx: 0.005 * (
            (pax4**3 + pax4) / (pax4**3 + 1)) * (
            (arx**3 + 2) / (arx**3 + arx + 1))

        beta_cells_prob = lambda mafa, delta: 0.003 * (
            (mafa**4 + mafa) / (mafa**4 + 1)) * (
            (delta**4 + 1) / (delta**4 + delta**2 + 1))
```

Using NumPy's *random.uniform* method, a million random densities for each of the six genes are generated. Then, probabilities of genes winning a switch is computed using the functions written above. Finally, NumPy's *random.choice* is used to find the winner according to the probabilites which are computed. A 1 means that the switch goes toward the creation beta cells.

In the Beta/Delta and Beta switches, the probabilty of that switch going toward beta cell creation is multiplied by probability of ending up in that switch.

```
In [3]: def markov_toar():
            # Endocrine Progenitor Sampling
            ptf1a_densities = np.random.uniform(low=0, high=1, size=10**6)
            ngn3_densities = np.random.uniform(low=0, high=1, size=10**6)
```

```
ngn3_win_probs = [
    endocrine_progenitor_prob(ngn3, ptf1a) for ngn3, ptf1a in zip(ptf1a_densities, n

endoc_prog_outcomes = [
    np.random.choice(
        [0, 1], p=[1-ngn3_win_prob, ngn3_win_prob]) for ngn3_win_prob in ngn3_win_pr

endoc_prog_prob = sum(endoc_prog_outcomes) / 1e6
print(f'Number of Ngn3 wins in a million samples without considering prior probabili
print(f'Probabilty of creation of a endocrine progenitor cell: {endoc_prog_prob}')
print('')

# Beta/Delta Progenitor Sampling
pax4_densities = np.random.uniform(low=0, high=1, size=10**6)
arx_densities = np.random.uniform(low=0, high=1, size=10**6)

pax4_win_probs = [
    beta_delta_progenitor_prob(pax4, arx) for pax4, arx in zip(
        pax4_densities, arx_densities)]

beta_delta_win_probs = [
    pax4_win_prob for pax4_win_prob in pax4_win_probs]

beta_delta_outcomes = [
    np.random.choice(
        [0, 1], p=[1-beta_delta_win_prob, beta_delta_win_prob]) for beta_delta_win_p

# the probabilty of Pax4 winning the switch is multiplied by
# the probabilty of endocrine progenitor winning the previous switch.
beta_delta_prob = sum(beta_delta_outcomes) / 1e6 * endoc_prog_prob

print(f'Number of Pax4 wins in a million samples without considering prior probabili
print(f'Probabilty of creation of a Beta/Delta progenitor cell: {beta_delta_prob}')
print('')

# Beta Cell Sampling
mafa_densities = np.random.uniform(low=0, high=1, size=10**6)
delta_densities = np.random.uniform(low=0, high=1, size=10**6)

mafa_win_probs = [
    beta_cells_prob(mafa, delta) for mafa, delta in zip(
        mafa_densities, delta_densities)]


beta_cell_win_probs = [
    mafa_win_prob for mafa_win_prob in mafa_win_probs]
```

```
        beta_cell_outcomes = [
            np.random.choice(
            [0, 1], p=[1-beta_cell_win_prob, beta_cell_win_prob]) for beta_cell_win_prob in

        # the probabilty of Mafa winning the switch is multiplied
        # by the probabilty of Ngn3 and Pax4 winning the previous switches.
        beta_cell_prob = sum(beta_cell_outcomes) / 1e6 * beta_delta_prob

        print(f'Number of Mafa wins in a million samples without considering prior probabili
        print(f'Probabilty of creation of a Beta cell: {beta_cell_prob}')
        print('')
```

```
In [4]: %%time
        markov_toar()
```

```
Number of Ngn3 wins in a million samples without considering prior probabilites: 420
Probabilty of creation of a endocrine progenitor cell: 0.00042

Number of Pax4 wins in a million samples without considering prior probabilites: 3630
Probabilty of creation of a Beta/Delta progenitor cell: 1.5246e-06

Number of Mafa wins in a million samples without considering prior probabilites: 1284
Probabilty of creation of a Beta cell: 1.9575864e-09

CPU times: user 1min 28s, sys: 1.35 s, total: 1min 30s
Wall time: 1min 32s
```