Hands-On Programming Sessions

Proteomes, Interactomes and Biological Networks

Emidio Capriotti

December 2, 2020

Internetional Degree in Genomics — University of Bologna (Italy)

Contents

1 Protein Sequence and Structure Data

Exercise 1: Handling Protein Sequence Data.

From the UniProt FTP web site (ftp://ftp.expasy.org/databases/uniprot/) download the Human protein UP000005640_9606 in fasta format.

- What is the total number of human proteins in the SwissProt and TrEMBL dataset?
- Given the fasta file containing the protein sequence of P53 what is the tota number of residues?

[ ]:
```
# For downloading the human proteome file
!wget ftp://ftp.expasy.org/databases/uniprot/current_release/knowledgebase/
 ↪reference_proteomes/Eukaryota/UP000005640_9606.fasta.gz
```

[3]:
```
# Total numer of SwissProt protein
!echo -e "Annotated Human Protein: \c"
!zcat data/UP000005640_9606.fasta.gz |grep ">sp" | wc -l

# Total number of TrEMBL protein
!echo -e "Human Protein TrEMBL: \c"
!zcat data/UP000005640_9606.fasta.gz |grep ">tr" | wc -l
```

```
Annotated Human Protein: 20296
Human Protein TrEMBL: 304
```

[ ]:
```
# Download P53 file sequence
!wget https://www.uniprot.org/uniprot/P04637.fasta
```

[5]:
```
# Count residues
!echo -e "P53 Sequence Length: \c"
!grep -v ">" data/P04637.fasta | tr -d "\n" |wc -c
```

```
P53 Sequence Length: 393
```

Exercise 2: Handling Protein Structure Data.

Download the PDB file of the Ribonuclease A (PDB: 7RSA) from the web (http://ftp.rcsb.org/pub/pdb/data/structures/all/pdb/pdb7rsa.ent.gz) and perform the following tasks * Run a shell command to calculate the number of residues of the protein? * Write a python script to parse the PDB file. * Modify the program to calculate the distance between to atoms and residues. * Calculate the average and standard deviation of the distance between two consecutive α-carbons

[7]:
```
# Download the PDB file
!wget http://ftp.rcsb.org/pub/pdb/data/structures/all/pdb/pdb7rsa.ent.gz

# Count the CA atmes
!echo -e "Residues in PDB Structure: \c"
!zcat data/pdb7rsa.ent.gz |grep ^ATOM |grep -w CA |wc -l
```

```
Residues in PDB Structure: 124
```

[ ]:
```python
#!/usr/bin/env python3
import sys
import numpy as np

# Function parsing the PDB file
def parse_pdb(filename, chain):
  pdb_coords={}
  f=open(filename,'r')
  for line in f:
      line=line.rstrip()
      # Check the field ATOM at the beginning of the line
      # Check the correct chain in column 22
      if line[:4]!='ATOM' or line[21]!=chain: continue
      resn=int(line[22:26].strip())
      atom=line[12:16].strip()
      x=float(line[30:38])
      y=float(line[38:46])
      z=float(line[46:54])
      coord=[x,y,z]
      # Initialize the dictionary with atoms coordinates of residue resn
      pdb_coords[resn]=pdb_coords.get(resn,{})
      # Add the atom's coordinates
      pdb_coords[resn][atom]=coord
  return pdb_coords


# Function calculating the distance between to points
def get_distance(coord1,coord2):
  return np.sqrt((coord1[0]-coord2[0])**2+\
                 (coord1[1]-coord2[1])**2+\
                 (coord1[2]-coord2[2])**2)



# Function returning the diatnces between all consecutives CA
def get_ca_dist(pdb_coords):
  ca_dists=[]
  keys=list(pdb_coords.keys())
  keys.sort()
  n=len(keys)
  for i in range(n-1):
    dist=get_distance(pdb_coords[keys[i]]['CA'],pdb_coords[keys[i+1]]['CA'])
    ca_dists.append(dist)
  return ca_dists


if __name__ == '__main__':
  filename=sys.argv[1]
  chain=sys.argv[2]
  pdb_coords=parse_pdb(filename,chain)
  ca_dists=get_ca_dist(pdb_coords)
  print ('Dist CA:',np.mean(ca_dists),np.std(ca_dists))))
```

[ ]:
```
# Run the script calculating the distance between the firt two CA atoms
!python3 script/parse_pdb.py data/pdb7rsa.ent A
```

```
Dist CA: 3.785132098091162
```

Exercise 3: Analysis of the secondary structure of protein from DSSP file.

Download the DSSP file of the Ribonuclease A (PDB: 7RSA) from the web (ftp://ftp.cmbi.umcn.nl//pub/molbio/data/dssp/7rsa.dssp) and answer the following questions:

- What is the total number of residues in helical and extended conformations?
- What is the average value of the and angles for the residues in helical and extended conformations?
- Are the average values falling the the correct region of the Ramachandran plot?
- Considering the solvent accessibility values reported in the DSSP file, calculate the relative solvent accessible area for Lysine, Valine and Glutamine with maximum solvent accessible area of 205, 142 and 198 respectively
- Are this value compatible with the physico-chemical properties of the residues?

[ ]:
```
# Download the DSSP file
wget ftp://ftp.cmbi.umcn.nl//pub/molbio/data/dssp/7rsa.dssp
```

[ ]:
```
#!/usr/bin/env python3
import sys
import numpy as np

# Function parsing the DSSP file
def parse_dssp(filename,chain):
  dssp_list=[]
        # Initialize state variable c=0
  c=0
  f=open(filename)
  for line in f:
    line=line.rstrip()
                # Check for the beginning of data
                # Change the state variable c=1
    if line.find('  #  RESIDUE')>-1:
      c=1
      continue
                # Check if the state variable is 1
                # Select the correct chain
    if c==0 or line[11]!=chain: continue
                # Read all the variables resn, aa, acc, phi, psi
```

```python
    aa=line[13]
    if aa.islower(): aa='C
                if aa=='!': continue
    resn=int(line[5:10])
    ss=line[16]
    if ss==' ': ss='C'
    acc=float(line[34:38])
    phi=float(line[103:109])
    psi=float(line[109:115])
    aa_dssp=[resn,aa,ss,acc,phi,psi]
    dssp_list.append(aa_dssp)
  return dssp_list


# Get sequence and secondary structure
def get_ss(dssp_list):
        seq=''
        ss=''
        for aa in dssp_list:
                ss=ss+aa[2]
                seq=seq+aa[1]
        return seq,ss


# Count residues with a given secondary structure
def count_ss(ss,ss_type):
        c=0
        for i in ss:
                if i==ss_type: c=c+1
        return c


# Return the list of all phi and psi angles
def get_ss_angle(dssp_list,ss_type):
        angles=[]
        for aa in dssp_list:
                if aa[2]==ss_type: angles.append((aa[-2],aa[-1]))
        return angles


# Return the list of residues accessibility
def get_aa_acc(dssp_list,aa_type):
        accs=[]
        for aa in dssp_list:
                if aa[1]==aa_type: accs.append(aa[3])
        return accs
```

```python

def get_chain_acc(dssp_list):
        accs=[]
        for aa in dssp_list:
                accs.append(aa[3])
        return accs



if __name__ == '__main__':
        filename=sys.argv[1]
        chain=sys.argv[2]
        dssp_list=parse_dssp(filename,chain)
        seq,ss=get_ss(dssp_list)
        print ('>SEQ\n%s' %seq)
        print ('>SS\n%s' %ss)
        print ('H count:',count_ss(ss,'H'))
        print ('E count:',count_ss(ss,'E'))
        h_angles=get_ss_angle(dssp_list,'H')
        e_angles=get_ss_angle(dssp_list,'E')
        print ('H angles:',np.mean([i[0] for i in h_angles]),np.mean([i[1] for␣
 ↪i in h_angles]))
        print ('E angles:',np.mean([i[0] for i in e_angles]),np.mean([i[1] for␣
 ↪i in e_angles]))
        print ('K acc:',np.mean(get_aa_acc(dssp_list,'K'))/205)
        print ('V acc:',np.mean(get_aa_acc(dssp_list,'V'))/142)
```

[ ]:
```python
# Run the script calculating the distance between the firt two CA atoms
!python3 script/parse_dssp.py data/7rsa.dssp A
```

```
>SEQ
KETAAAKFERQHMDSSTSAASSSNYCNQMMKSRNLTKDRCKPVNTFVHESLADVQAVCSQKNVACKNGQTNCYQSYSTMS
ITDCRETGSSKYPNCAYKTTQANKHIIVACEGNPYVPVHFDASV
>SS
CCCHHHHHHHHHBCTTCSSCCSTTHHHHHHHHTTTTSSSCCSEEEEECSCHHHHHGGGGSEEECCTTSCSCEEECSSCEE
EEEEEECTTCBTTBCCEEEEEEEEECEEEEEETTTTEEEEEEEEC
H count: 22
E count: 41
H angles: -65.83636363636364 -41.00454545454546
E angles: -117.89268292682925 132.09756097560978
K acc: 0.6404878048780488
V acc: 0.18309859154929578
```

2 Amino Acid Propensities

Exercise 1: Generate a secondary structure propensity scale.

Develop your own alpha helix propensity scale based on the data available at the link http://biofold.org/pages/courses/pibn/data/ss_data.tsv.gz.

- Compare your scale with the AAindex Chou-Fasman scale
- Write a script that given a sequence and propensity scale calculates the smoothed score on a window sequence.

[ ]:
```
# For downloading the secondary structure data file
!wget http://biofold.org/pages/courses/pibn/data/ss_data.tsv.gz
```

[ ]:
```
# Unzip the file
!gunzip ss_data.tsv.gz
```

[ ]:
```
#!/usr/bin/env python3
import sys
import numpy as np


# Parsing secondary structure file containing 4 columns
# PDBID CHAIN SEQUENCE SECONDARY_STRUCTURE
def parse_ssdata(filename):
        # Parsing ss_data and generete 3 dicitionaries
        # The dicitionaries are for counting the AA, SS and (AA,SS)
        d_aa={}
        d_ss={}
        d_aass={}
        f=open(filename,'r')
        for line in f:
                line=line.rstrip()
                cols=line.split('\t')
                if (len(cols[2])!=len(cols[3])): print ('WARNING: Incorrect␣
 ↪data.',file=sys.stderr)
                n=len(cols[2])
                for i in range(n):
                        aa=cols[2][i]
                        ss=cols[3][i]
                        d_aa[aa]=d_aa.get(aa,0)+1
                        d_ss[ss]=d_ss.get(ss,0)+1
                        d_aass[(aa,ss)]=d_aass.get((aa,ss),0)+1
        return d_aa,d_ss,d_aass


# Calculate the propensity scale for a given SS
def calculate_ssprop(d_aa,d_ss,d_aass,ss_type):
        d_prop={}
        aas=list(d_aa.keys())
        n=float(sum(list(d_aa.values())))
        for aa in aas:
                # Check for zero countings
                if d_aass.get((aa,ss_type),0)==0 or \
                        d_aa.get(aa,0)==0 or \
                        d_ss.get(ss_type,0)==0: continue
                d_prop[aa]=(d_aass[(aa,ss_type)]/n)/(d_aa[aa]/n*d_ss[ss_type]/n)
        return d_prop


if __name__ == '__main__':
        filename=sys.argv[1]
        ss_type=sys.argv[2]
        d_aa,d_ss,d_aass=parse_ssdata(filename)
        d_prop=calculate_ssprop(d_aa,d_ss,d_aass,ss_type)
  ks=list(d_prop.key())
  for k in ks:
                print ('%s\t%s:\t%.2f' %(ss_type,k,d_prop[k]))
```

[11]:

```
# Run the script calculating the popensity scale
!python3 script/ss_scale.py data/ss_data.tsv H
```

```
H       R:      1.23
H       S:      0.74
H       P:      0.43
H       A:      1.44
H       D:      0.84
H       I:      1.10
H       K:      1.16
H       V:      0.92
H       H:      0.70
H       T:      0.77
H       E:      1.38
H       G:      0.44
H       L:      1.35
H       Q:      1.29
H       N:      0.72
H       M:      1.15
H       C:      0.86
H       Y:      1.00
H       F:      1.00
H       W:      1.07
H       X:      1.05
H       U:      0.60
```

```
H       B:      3.28
```

3 Protein-Protein Interaction

Exercise 1: Analysis of the protein complex 1BRL.

Download the DSSP file of the Bacterial luciferase (*Vibrio harveyi*) from the PDB (code: 1BRL)

- Generate the PDB file for the protein complex and the isolated chains A and B
- Calculate the total solvent accessible area of the complex and isolated chains and calculate the surface of interaction for both chains.
- Given the size of the binding surface what kind of protein interaction it is expected?
- Find the residue at the interface and calculate the variation of relative solvent accessible area. Which residue are buried in the interacting surface?

[ ]:
```
# Download the PDB file
!wget http://ftp.rcsb.org/pub/pdb/data/structures/all/pdb/pdb1brl.ent.gz
!mv pdb1brl.ent.gz data

# Unzip the pdb file
!gunzip data/pdb1brl.ent.gz

# Extract the chain A
!grep -w A data/pdb1brl.ent >data/1brlA.pdb

# Extract the chain B
!grep -w B data/pdb1brl.ent >data/1brlB.pdb

# Merge chains A and B
!cat data/1brlA.pdb data/1brlB.pdb >data/1brlAB.pdb

# Generate the DSSP file for chain A
!script/dssp data/1brlA.pdb >data/1brlA.dssp

# Generate the DSSP file for chain B
!script/dssp data/1brlB.pdb >data/1brlB.dssp

# Generate the DSSP file for the complex AB
!script/dssp data/1brlAB.pdb >data/1brlAB.dssp
```

Modify the previous parse_dssp.py script to print in output the accessibility of each residue and the total accessibility. Rename the script parse_dssp_ppi.py

[ ]:
```
#!/usr/bin/env python3
import sys
import numpy as np

# Function parsing the DSSP file
def parse_dssp(filename,chain):
  dssp_list=[]
```

```
    # Initialize state variable c=0
 c=0
 f=open(filename)
 for line in f:
  line=line.rstrip()
        # Check for the beginning of data
        # Change the state variable c=1
  if line.find('  #  RESIDUE')>-1:
   c=1
   continue
        # Check if the state variable is 1
        # Select the correct chain
  if c==0 or line[11]!=chain: continue
        # Read all the variables resn, aa, acc, phi, psi
  aa=line[13]
  if aa.islower(): aa='C
        if aa=='!': continue
  resn=int(line[5:10])
  ss=line[16]
  if ss==' ': ss='C'
  acc=float(line[34:38])
  phi=float(line[103:109])
  psi=float(line[109:115])
  aa_dssp=[resn,aa,ss,acc,phi,psi]
  dssp_list.append(aa_dssp)
 return dssp_list


# Get sequence and secondary structure
def get_ss(dssp_list):
    seq=''
    ss=''
    for aa in dssp_list:
        ss=ss+aa[2]
        seq=seq+aa[1]
    return seq,ss


# Count residues with a given secondary structure
def count_ss(ss,ss_type):
    c=0
    for i in ss:
        if i==ss_type: c=c+1
    return c


# Return the list of all phi and psi angles
```

```
def get_ss_angle(dssp_list,ss_type):
        angles=[]
        for aa in dssp_list:
                if aa[2]==ss_type: angles.append((aa[-2],aa[-1]))
        return angles


# Return the list of residues accessibility
def get_aa_acc(dssp_list,aa_type):
        accs=[]
        for aa in dssp_list:
                if aa[1]==aa_type: accs.append(aa[3])
        return accs



def get_chain_acc(dssp_list):
        accs=[]
        for aa in dssp_list:
                accs.append(aa[3])
        return accs




if __name__ == '__main__':
        filename=sys.argv[1]
        chain=sys.argv[2]
        dssp_list=parse_dssp(filename,chain)
        accs=get_chain_acc(dssp_list)
         for i_dssp in dssp_list:
                 print ('\t'.join([str(i) for i in i_dssp]))
        print ('TOTAL:',sum(accs))
```

[ ]:
```
# Run the script on the 1brlA.dssp file and redirect the output
!python3 script/parse_dssp_ppi.py data/1brlA.dssp A >data/m1brlA.acc

# Run the script on the 1brlB.dssp file and redirect the output
!python3 script/parse_dssp_ppi.py data/1brlB.dssp B >data/m1brlB.acc

# Run the script on the 1brlAB.dssp file and get the accessibility of chain A
!python3 script/parse_dssp_ppi.py data/1brlAB.dssp A >data/c1brlA.acc

# Run the script on the 1brlAB.dssp file and get the accessibility of chain B
!python3 script/parse_dssp_ppi.py data/1brlAB.dssp B >data/c1brlB.acc
```

[ ]:
```
# Show the Total surface of the monomer A
!tail -n 1 data/m1brlA.acc
```

```
# Show the Total surface of the monomer B
!tail -n 1 data/m1brlB.acc

# Show the Total surface of the chain A in complex
!tail -n 1 data/c1brlA.acc

# Show the Total surface of the chain B in complex
!tail -n 1 data/c1brlB.acc
```

```
TOTAL: 16527.0
TOTAL: 14529.0
TOTAL: 14330.0
TOTAL: 12338.0
```

The surface of interaction is:

$$InteractionSurface = [S(A) + S(B) + S(AB)]/2$$

where S(AB) is the sum of S(A) + S(B) obtaind for the structure of the protein complex (1br-lAB.pdb)

```
[ ]: # The surface of interaction is S(A) + S(B) + S(AB)
     # S(AB) is the sum of S(A) + S(B) obtaind for the
     # structure of the protein complex (1brlAB.pdb)

     (16527.0+14529.0-(14330.0+12338.0))/2.
```

```
[ ]: 2194.0
```

To find the residue at the interface we need to compare the dssp files of chains in the monomeric and complex forms. For this task we can use the command *paste* in conjunctionb with *awk*.

```
[ ]: # Compare chains A in monomeric and complex forms showing only residues with␣
      ↪differecne in accessibility greater than 50
     !paste data/m1brlA.acc data/c1brlA.acc |awk -v OFS='\t' '{if ($4-$10>50) print␣
      ↪$1,$2,$4,$10,$4-$10}' |sort -nrk 5  |grep -v TOTAL
```

```
85      R       180.0   25.0    155
96      M       130.0   7.0     123
54      N       107.0   3.0     104
61      H       118.0   18.0    100
57      V       103.0   3.0     100
46      F       114.0   21.0    93
17      Q       116.0   26.0    90
159     N       88.0    6.0     82
88      E       84.0    5.0     79
116     V       82.0    3.0     79
157     Q       119.0   43.0    76
21      M       72.0    0.0     72
```

```
117     F       84.0    15.0    69
92      L       68.0    1.0     67
95      Q       73.0    13.0    60
265     D       94.0    37.0    57
64      G       63.0    11.0    52
18      T       97.0    45.0    52
270     Y       215.0   164.0   51
```

[ ]:
```
# Compare chains B in monomeric and complex forms showing only residues with␣
↪differecne in accessibility greater than 50
!paste data/m1brlB.acc data/c1brlB.acc |awk -v OFS='\t' '{if ($4-$10>50) print␣
↪$1,$2,$4,$10,$4-$10}' |sort -nrk 5 |grep -v TOTAL
```

```
96      M       156.0   14.0    142
85      R       145.0   27.0    118
61      F       130.0   12.0    118
116     F       135.0   21.0    114
46      F       139.0   33.0    106
159     N       99.0    4.0     95
57      T       91.0    0.0     91
154     P       105.0   24.0    81
88      E       75.0    1.0     74
92      L       72.0    0.0     72
95      Q       70.0    6.0     64
21      I       64.0    4.0     60
64      G       71.0    13.0    58
117     F       64.0    6.0     58
```

4 Analysis of Protein-Protein Interaction Network

Exercise 1: Analysis of the protein-protein interactions in the IntAct database.

Using basic linux command and *awk* analyze the interactions collected in the IntAct database.

- Search for the interactions of the MEKK1 protein. How many interaction you can find? Are all this referring to the same protein?
- Refine your search using the UniProtID Q13233. How many interaction you have now?
- Search the interaction between BRAF and other proteins. Is BRAF interacting with MEKK1? How many experimental data are supporting the existence of this interaction?

- Extract from IntAct all the interactions between human proteins from UniProt. How many unique interactions are present?

[ ]:
```
# Download the IntAct PSI-MITAB file
!wget ftp://ftp.ebi.ac.uk/pub/databases/intact/current/psimitab/intact.zip
!mv intact.zip data

# Search for the inteactions of MEKK1 protein using grep and count them
!echo -e "Number of interactions of MEKK1: \c"
!unzip -c data/intact.zip |grep 'uniprotkb:MEKK1' > data/mekk1.txt
!cat data/mekk1.txt | wc -l

# From the previous search, count the number of unique interactions
!echo -e "Unique interactions of MEKK1: \c"
!cat data/mekk1.txt | awk -F '\t' '{print $1"\t"$2}' |sort -u |wc -l

# Remove interactions with drugs (chebi:) and other non proteins (intact:)
!echo -e "Unique interactions of MEKK1 with proteins: \c"
!cat data/mekk1.txt | awk -F '\t' '{print $1"\t"$2}' | grep -v 'chebi:' | grep␣
↪-v 'intact:' | sort -u | wc -l

# Select only interactions correponiding to the human MEKK1 searching for the␣
↪swissprot identifier Q13233
!echo -e "Unique interactions of the human MEKK1 with proteins: \c"
!grep  'uniprotkb:Q13233' data/mekk1.txt | awk -F '\t' '{print $1"\t"$2}' |␣
↪grep -v 'chebi:' | grep -v 'intact:' | sort -u | wc -l

# Calculate the number of inteaction of BRAF (P15056)
!echo -e "Unique interactions of BRAF with proteins: \c"
!unzip -c data/intact.zip |grep 'uniprotkb:P15056' | awk -F '\t' '{print␣
↪$1"\t"$2}' >data/braf.txt
!sort -u data/braf.txt | wc -l

# Is BRAF inteacting with MEKK1? How many experiments are supporting this␣
↪interactions?
!echo -e "Number of interactions between MEKK1 and BRAF: \c"
```

```
!cat data/braf.txt |grep 'uniprotkb:P15056' | grep  'uniprotkb:Q13233' | wc -l
```

```
Number of interactions of MEKK1: 180
Unique interactions of MEKK1: 158
Unique interactions of MEKK1 with proteins: 153
Unique interactions of the human MEKK1 with proteins: 132
Unique interactions of BRAF with proteins: 109
Number of interactions between MEKK1 and BRAF: 2
```

Now focus on a specific organism and search all the **protein-protein intaractions** between proteins of the **human proteome**.

```
[ ]: # Using the TaxID of the Homo sapiens (9606) calculate the numeber of total␣
     ↪interactions between human proteins
     !unzip -c data/intact.zip | awk -F '\t' '{split($10,org1,"|");␣
     ↪split($11,org2,"|"); if (org1[1]=="taxid:9606(human)" && org2[1]=="taxid:
     ↪9606(human)") print $0}' >data/human.txt
     !echo -e "Number of interactions between human proteins: \c"
     !cat data/human.txt | wc -l

     # What is the numeber of unique interactions considering the problem of the␣
     ↪directionality A->B is equal to B->A
     !echo -e "Unique interactions between human proteins: \c"
     !awk -F '\t' '{ if ($1>$2) {print $2"\t"$1} else {print $1"\t"$2} }' data/human.
     ↪txt | sort -u | wc -l
```

```
Number of interactions between human proteins: 584646
Unique interactions between human proteins: 302762
```

5 Introduction to Graph Theory and Network

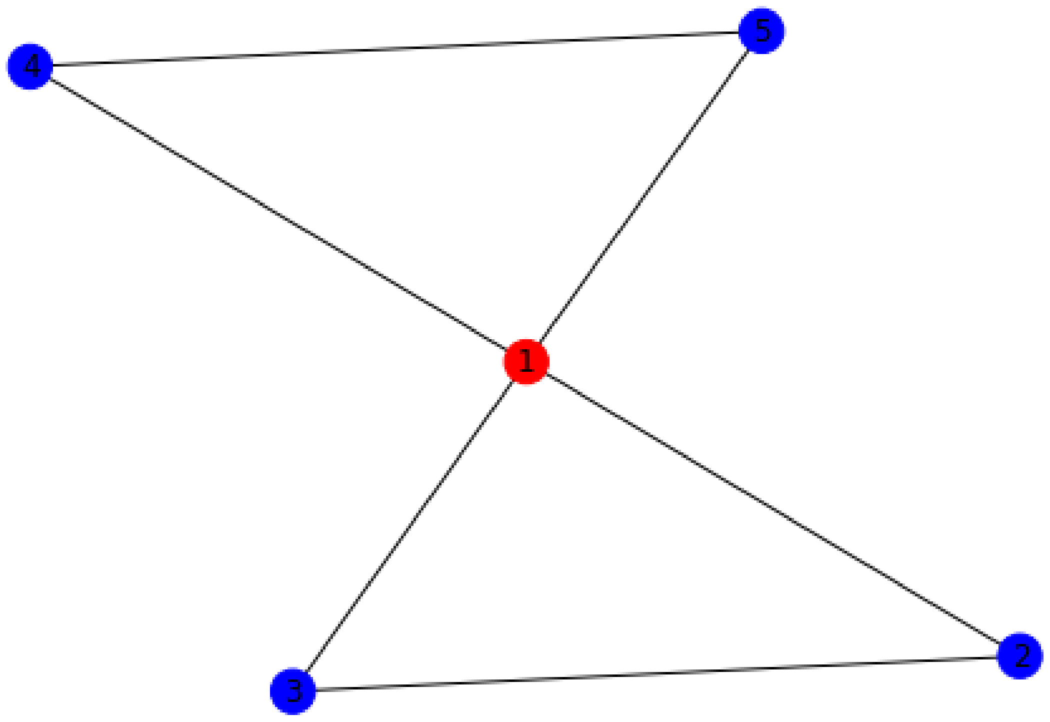
Exercise 1: Analysis of the graph topology

Calculate the degree and the clustering coefficient of node 1 in the following graph:

- *N=[1,2,3,4,5]*
- *E=[(1,2),(1,3),(1,4),(1,5),(2,3),(4,5)]*

[ ]:
```
import networkx as nx
import matplotlib.pyplot as plt

g=nx.Graph()
g.add_edges_from([(1,2),(1,3),(1,4),(1,5),(2,3),(4,5)])
c1=nx.clustering(g,1)
d1=g.degree(1)
print ('D(1)= %d' %d1)
print ('C(1)= %.3f' %c1)
colors=['red']+4*['blue']
nx.draw(g,with_labels=True,node_color=colors)
plt.show()
```

```
D(1)= 4
C(1)= 0.333
```

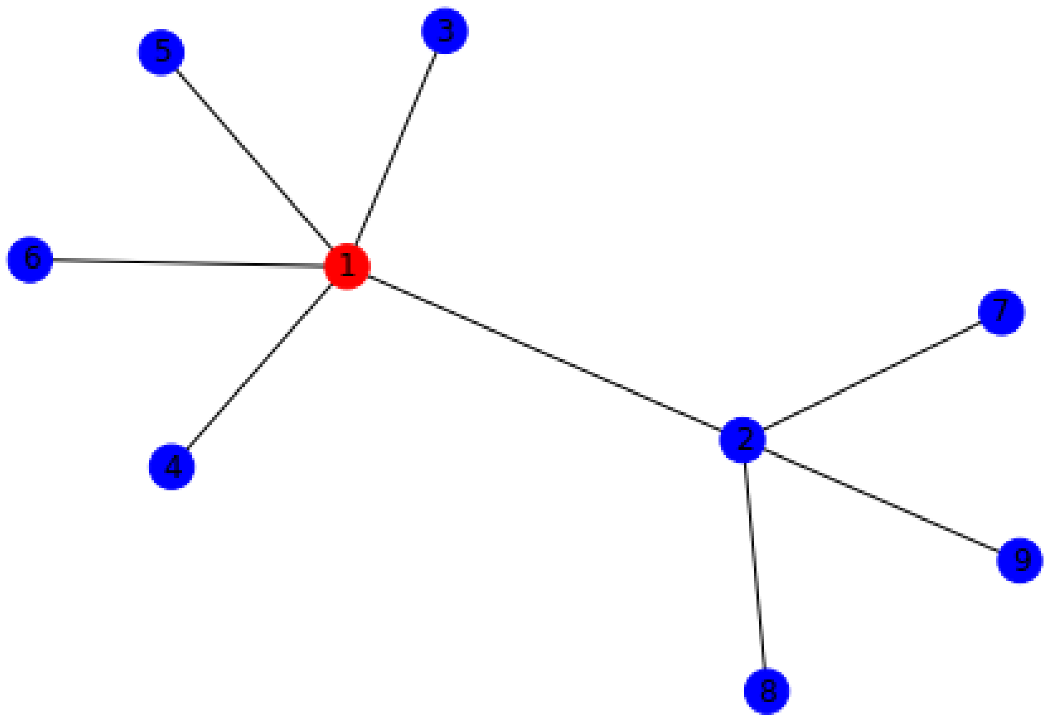


Calculate the degree and the numbero of shortest path of node 1 in the following graph:

- *N=[1,2,3,4,5,6,7,8,9]*
- *E=[(1,2),(1,3),(1,4),(1,5),(1,6),(2,7),(2,8),(2,9)]*

[ ]:
```
%matplotlib inline
import networkx as nx
import matplotlib.pyplot as plt

g=nx.Graph()
g.add_edges_from([(1,2),(1,3),(1,4),(1,5),(1,6),(2,7),(2,8),(2,9)])
s1=nx.betweenness_centrality_source(g,normalized=False)[1]
d1=g.degree(1)
print ('D(1)= %d' %d1)
print ('S(1)= %.0f' %s1)
colors=['red']+8*['blue']
nx.draw(g,with_labels=True,node_color=colors)
plt.show()
```

```
D(1)= 5
S(1)= 22
```



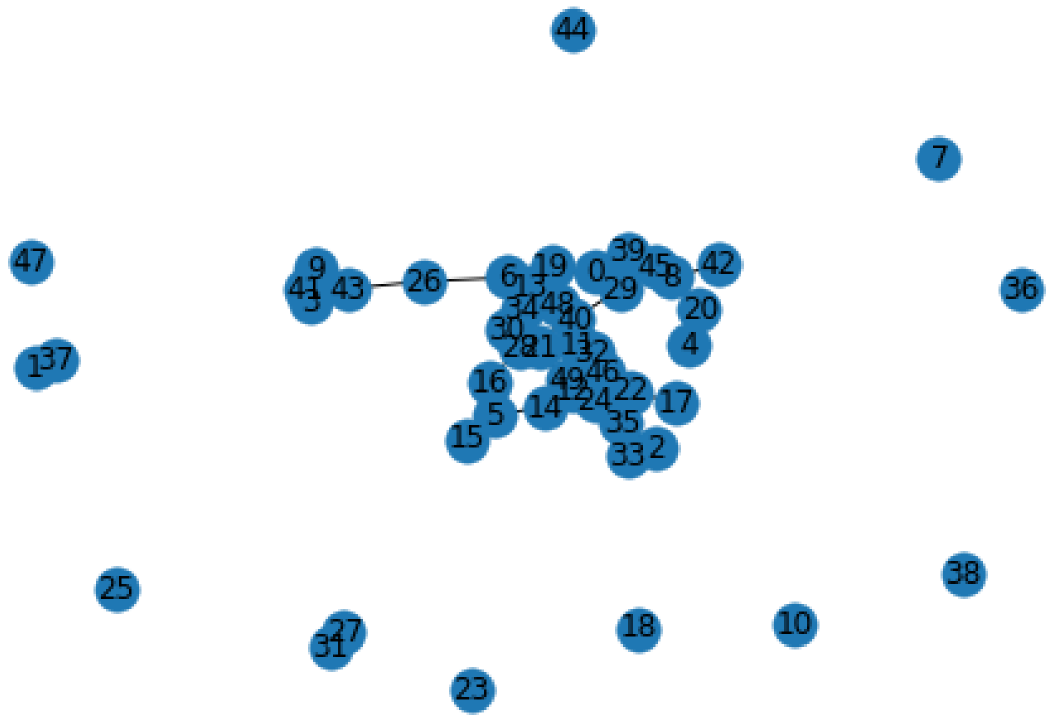
Exercise 2: Generate Random, Small-Word and Scale-Free networks and compare their topology

Use the networkx python library to generate the following networks:

1. Random network (Erdos Renyi model)
2. Small-world network (Watts-Strogatz model)
3. Scale-free network (Barabasi-Albert model)

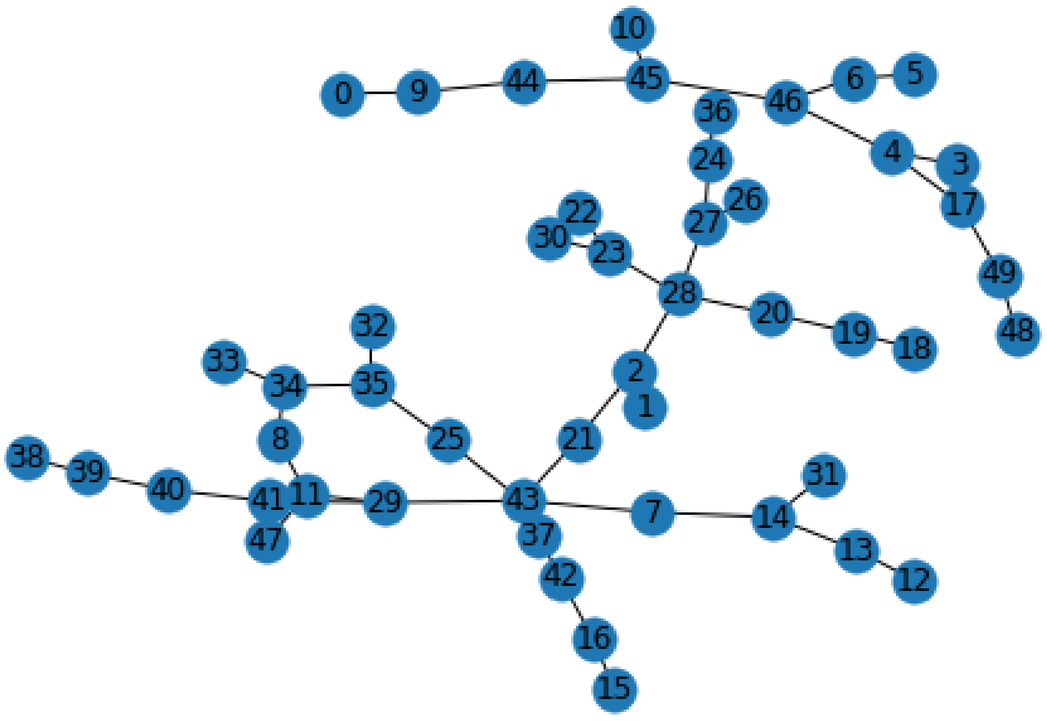
```
[ ]: import networkx as nx
     import matplotlib.pyplot as plt
     import numpy as np

     nodes=50
     er_net=nx.erdos_renyi_graph(nodes,0.04)
     mean_deg=np.mean(list(dict(er_net.degree()).values()))
     list_bets=list(nx.betweenness_centrality_source(er_net,normalized=False).
      ↪values())
     mean_bet=np.mean(list_bets)
     mean_tra=np.mean(list(nx.clustering(er_net).values()))
     print ('<DEGREE>:',mean_deg)
     print ('<BETWEENNESS>:',mean_bet)
     print ('<TRANSITIVITY>: %.3f' %mean_tra)
     nx.draw(er_net,with_labels = True)
     plt.show()
```

```
<DEGREE>: 1.96
<BETWEENNESS>: 44.02
<TRANSITIVITY>: 0.091
```



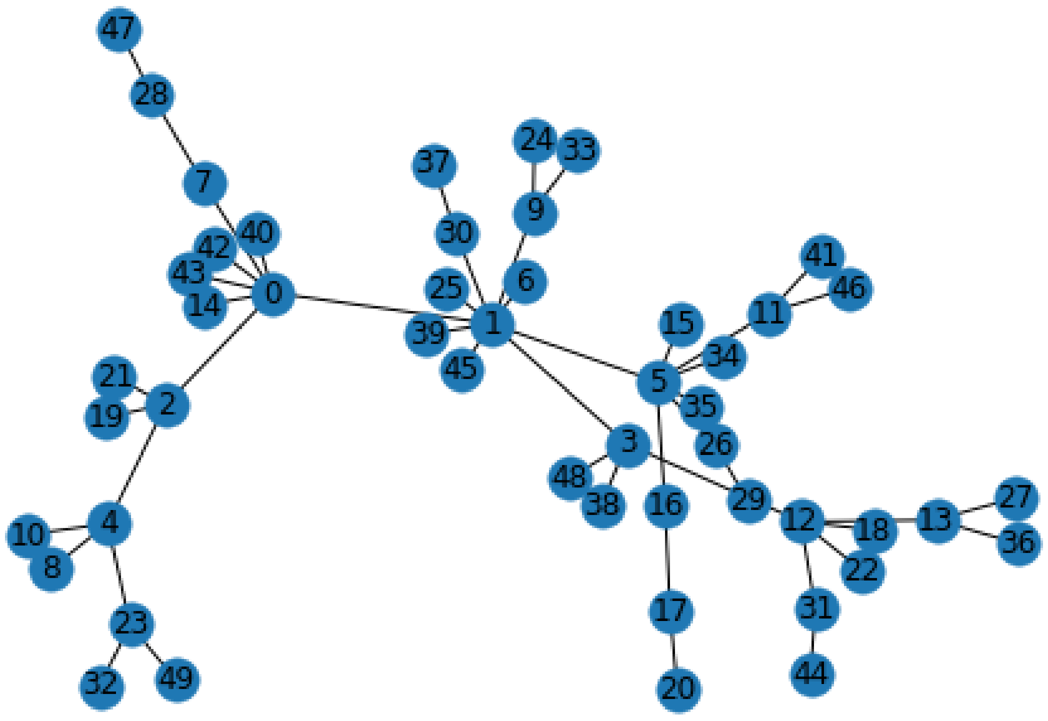
```
[ ]: import networkx as nx
     import matplotlib.pyplot as plt
     import numpy as np

     nodes=50
     ws_net=nx.watts_strogatz_graph(nodes,2,0.5)
     mean_deg=np.mean(list(dict(ws_net.degree()).values()))
     list_bets=list(nx.betweenness_centrality_source(ws_net,normalized=False).
      ↪values())
     mean_bet=np.mean(list_bets)
     mean_tra=np.mean(list(nx.clustering(ws_net).values()))
     print ('<DEGREE>:',mean_deg)
     print ('<BETWEENNESS>:',mean_bet)
     print ('<TRANSITIVITY>: %.3f' %mean_tra)
     nx.draw(ws_net,with_labels = True)
     plt.show()
```

```
<DEGREE>: 2.0
<BETWEENNESS>: 58.92
<TRANSITIVITY>: 0.033
```



```
[ ]: import networkx as nx
     import matplotlib.pyplot as plt
     import numpy as np

     nodes=50
     ba_net=nx.barabasi_albert_graph(nodes,1)

     mean_deg=np.mean(list(dict(ba_net.degree()).values()))
     list_bets=list(nx.betweenness_centrality_source(ba_net,normalized=False).
      ↪values())
     mean_bet=np.mean(list_bets)
     mean_tra=np.mean(list(nx.clustering(ba_net).values()))
     print ('<DEGREE>:',mean_deg)
     print ('<BETWEENNESS>:',mean_bet)
     print ('<TRANSITIVITY>: %.3f' %mean_tra)
     nx.draw(ba_net,with_labels = True)
     plt.show()
```

```
<DEGREE>: 1.96
<BETWEENNESS>: 81.68
<TRANSITIVITY>: 0.000
```



Exercise 3: Analysis of the Yeast interactome

Download from the BioGRID database the Yeast interactome in the psi-mitab format. Generate the network with undirected edges with networkx.

- How many components are present?
- What are their number of nodes and edges?
- What is the average values of degrees, clustering and betweenness?
- What is the gene with highest degree?
- Draw the distribution of the degree and fit to a power law distribution.

[ ]:
```
# Download the zip file of all interactomes
%env web=https://downloads.thebiogrid.org/Download/BioGRID/Release-Archive/
 ↪BIOGRID-4.2.191/BIOGRID-ORGANISM-4.2.191.mitab.zip
!wget $web -P data

# unzip only the Saccharomyces file
%env yeast=BIOGRID-ORGANISM-Saccharomyces_cerevisiae_S288c-4.2.191.mitab.txt
!unzip -p data/BIOGRID-ORGANISM-4.2.191.mitab.zip $yeast > data/$yeast
```

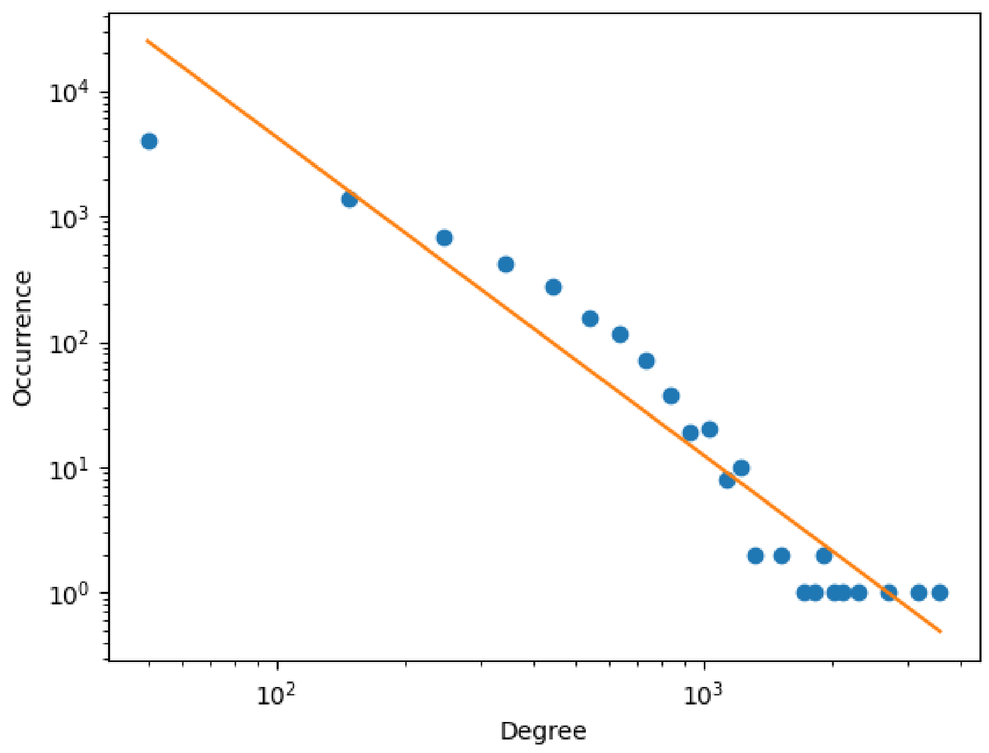
[ ]:
```
#!/usr/bin/env python3
import sys
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

def parse_mitab(filename,id1=2,id2=3):
        gi={}
        f=open(filename,'r')
        for line in f:
                if line[0]=='#': continue
                line=line.rstrip()
                v=line.split('\t')
                gid1=v[id1].split('|')[1].split(':')[1]
                gid2=v[id2].split('|')[1].split(':')[1]
                gids=[gid1,gid2]
                # sort gene identifiers to generate undirected edges
                gids.sort()
                gi[tuple(gids)]=True
        return list(gi.keys())


def get_net(edges):
        g=nx.Graph()
        g.add_edges_from(edges)
        return g


def print_net_features(net):
        # Search for components
        comps=nx.connected_components(net)
        i=1
        for comp in comps:
                print ('Component:',i)
                # Select subgraph
                sg=net.subgraph(comp)
                print ('\t#Nodes:',sg.number_of_nodes())
                print ('\t#Edges:',sg.number_of_edges())
                i=i+1
        # Calculate network features
        dgs=net.degree()
        print ('MEAN DEGREE:',np.mean(list(dgs)))
        cs=nx.clustering(net)
        print ('MEAN CLUSTERING:',np.mean(list(cs.values())))
        # Betweenness for all nodes is too long to be clalculated
        #bs=nx.betweenness_centrality_source(net,normalized=False)
        #print ('MEAN BETWEENNES: %.1f' %np.mean(list(bs.values())))
        # Find gene with highest degree
        list_dgs=[(v,k) for k,v in dgs.items()]
        list_dgs.sort()
        list_dgs.reverse()
        print ('MAX DEGREE:',list_dgs[0][1],list_dgs[0][0])



def fit_deg_distribution(degs_vals,nbin,outfile='figure'):
        hist,bins=np.histogram(degs_vals,nbin)
        # Transform the power law function in linear
        # calculating the log10 of x and y
        logx=[]
        logy=[]
        for i in range(len(hist)):
                if hist[i]==0: continue
                logy.append(np.log10(hist[i]))
                logx.append(np.log10((bins[i+1]+bins[i])/2))
        reg=linregress(logx,logy)
        print ('CORRELATION: %.3f' %reg[2])
        print ('p-value: %.3e' %reg[3])
        x=[]
        yf=[]
        y=[]
        # Calculate the inverse fuction to transform
        # the linear function to power law.
        for i in range(len(logx)):
                x.append(10**logx[i])
                y.append(10**logy[i])
                yf.append(10**(logx[i]*reg[0]+reg[1]))
        plt.plot(x,y,'o')
        plt.plot(x,yf)
        plt.xscale('log')
        plt.yscale('log')
        plt.savefig(outfile+'.png')


if __name__ == '__main__':
        filename=sys.argv[1]
        nbin=int(sys.argv[2])
        gi=parse_mitab(filename)
        net=get_net(gi)
        print_net_features(net)
        degs_vals=list(net.degree().values())
        fit_deg_distribution(degs_vals,nbin)
```

[ ]:
```
# Run the above script giving in input
# the mitab file and the number of bins.
!python3 script/biogrid-net.py <(zcat data/Saccharomyces_cerevisiae_S288c-4.2.
 ↪191.mitab.txt.gz) 37
```

```
Component: 1
        #Nodes: 7233
        #Edges: 547144
MEAN DEGREE: 151.3
MEAN CLUSTERING: 0.209
MAX DEGREE: DHH1 3622
CORRELATION: -0.954
p-value: 4.992e-13
```

[ ]:
```
from IPython.display import Image
Image('data/yeast.png')
```

[ ]:


10^4
10^3
10^2
10^1
10^0
Occurrence
10^2
10^3
Degree

6 Motif Matching and Statistical Analysis

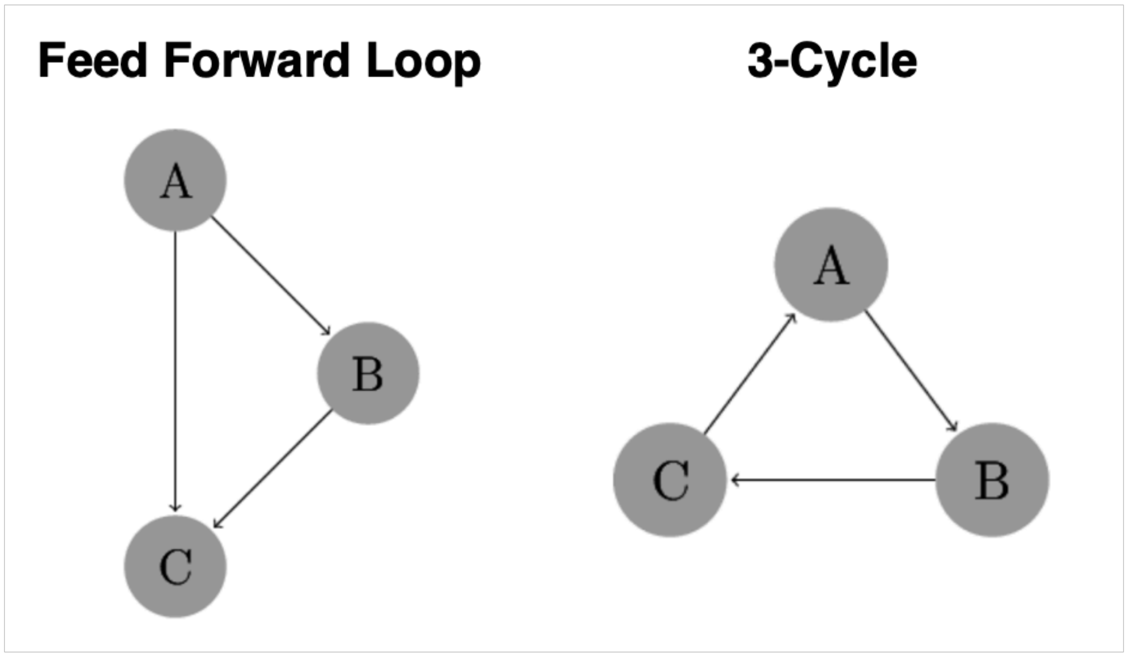
Exercise 1: Match motifs on a network

Given the Feed Forward Loop (FFL) and 3-Cycle motif:

- *Edges_FFL = [('A','B'),('A','C'),('B','C')]*
- *Edges_3C = [('A','B'),('B','C'),('C','A')]*

[ ]:
```
from IPython.display import Image
Image('data/motifs.png',width = 350)
```

[ ]:

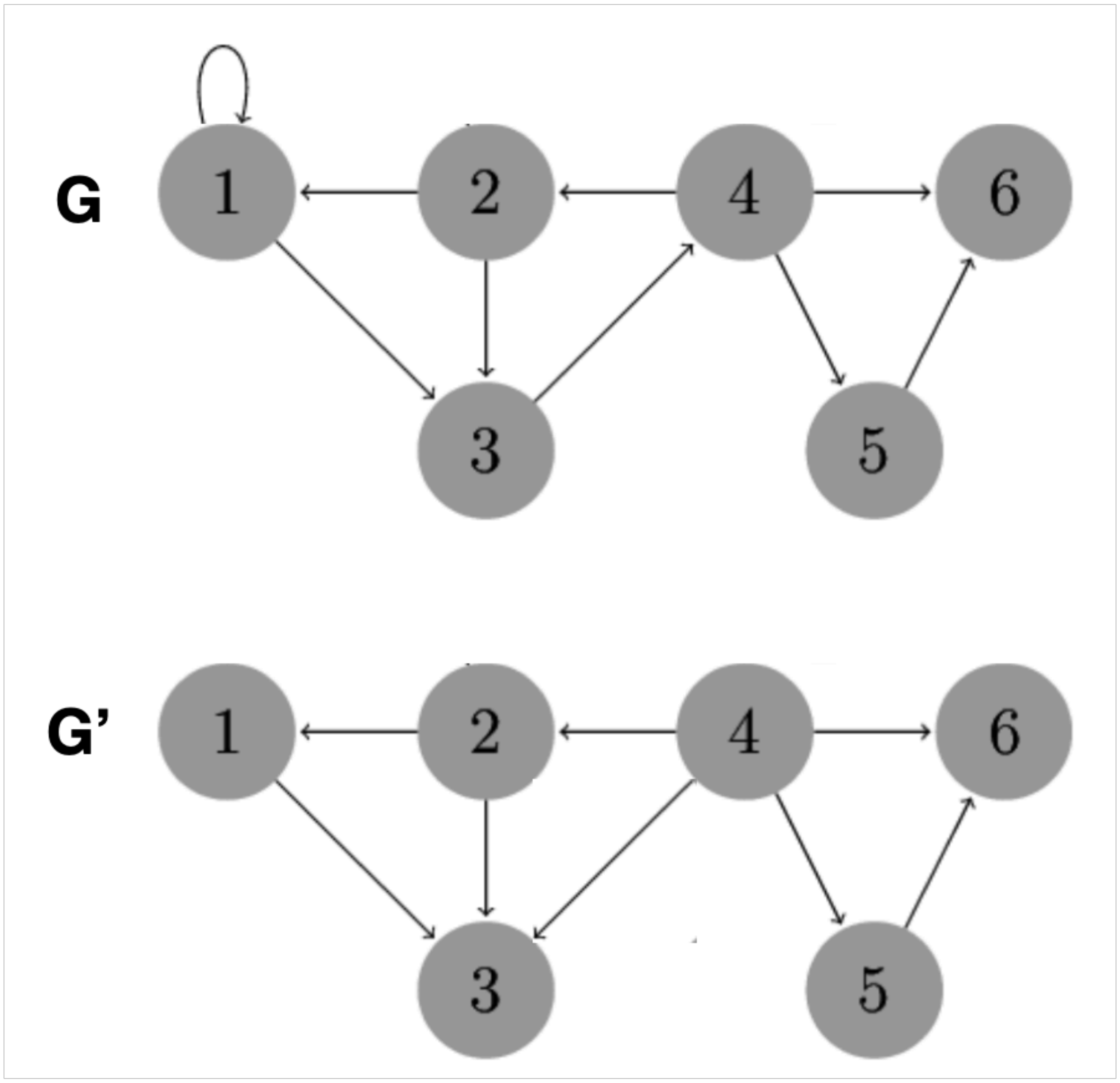


and the networks:

- *Edges_G = [(1,1), (1,3), (2,1), (2,3), (4,2), (4,3), (4,5), (4,6),(5,6)]*
- *Edges_G' = [(1,3), (2,1), (2,3), (3,4), (4,2), (4,5), (4,6), (5,6)]*

[ ]:
```
from IPython.display import Image
Image('data/nets.png',width = 350)
```

[ ]:



Write the code to:

- Match the motifs with the graph G
- Calculate occurrences on random networks and the z-score.
- What is the difference when the FFL is matched on G'?

[ ]:
```python
#!/usr/bin/env python3
import networkx as nx
import matplotlib.pyplot as plt
import itertools as it
import numpy as np
from scipy.stats import norm, shapiro
```

```
def count_isomorphisms(net,mot):
        n=list(net.nodes())
        k=mot.number_of_nodes()
        combs=list(it.combinations(n,k))
        i=0
        for comb in combs:
                sg=net.subgraph(comb)
                if nx.is_isomorphic(sg,mot): i=i+1
        return i


def match_isomorphisms(net,mot):
        diso={}
        ms=nx.isomorphism.DiGraphMatcher(net,mot)
        for m in ms.subgraph_isomorphisms_iter():
                k=list(m.keys())
                k.sort()
                diso[tuple(k)]=True
        i=len(list(diso.keys()))
        return i


def check_random(nn,ne,mot,k):
        occ=[]
        for i in range(k):
                g=nx.gnm_random_graph(nn,ne,directed=True)
                c=count_isomorphisms(g,mot)
                occ.append(c)
        return occ


def get_norm_statistics(net,mot,k=100):
        nn=net.number_of_nodes()
        ne=net.number_of_edges()
        # Here we used the maching function
        # based on DiGraphMatcher replacing
        # n_match=count_isomorphisms(net,mot)
        # with
        n_match=match_isomorphisms(net,mot)
        # Generate k random networks and
        # get their number of matches
        r_match=check_random(nn,ne,mot,k)
        # Calculate mean, stdev and z-score
        mean=np.mean(r_match)
        std=np.std(r_match)
        z=(n_match-mean)/std
        # Calculate the p-value assuming that
        # r_match values have a normal distribution
        p=norm.sf(n_match,mean,std)
        # Shapiro test for testing tha r_match values
        # have a normal distribution
        pshapiro=shapiro(r_match)[1]
        # Calculate Empirical probability
        ep=len([i for i in r_match if i>n_match])/float(len(r_match))
        print ('Matches:',n_match)
        print ('Z-score: %.3f' %z)
        print ('P-value %.3e' %p)
        print ('Shapiro: %.3e' %pshapiro)
        print ('P(t>%d): %.3e\n' %(n_match,ep))


def main():
        k=1000
        # Define Feed Forward Loop
        ffl=nx.DiGraph()
        ffl.add_edges_from([('A','B'),('A','C'),('B','C')])
        # Define 3-node Cycle
        c3=nx.DiGraph()
        c3.add_edges_from([('A','B'),('B','C'),('C','A')])
        # Define the network
        g=nx.DiGraph()
        g.
 ↪add_edges_from([(1,1),(2,1),(1,3),(2,3),(3,4),(4,2),(4,5),(4,6),(5,6)])
        print ('# Match Feed-Forvard Loop')
        get_norm_statistics(g,ffl,k)
        print ('# Match 3-Node Cycle')
        get_norm_statistics(g,c3,k)
        # Modify the network removing edge (1,1) and changing (3,4) with (4,3).
        g.remove_edge(1,1)
        g.remove_edge(3,4)
        g.add_edge(4,3)
         # Test the significance of the occurrence of the Feed-Forvard Loop
        print ('# Match Feed-Forvard Loop')
        get_norm_statistics(g,ffl,k)


if __name__ == '__main__':
        main()
```

```
# Match Feed-Forvard Loop
Matches: 1
Z-score: -0.111
P-value 5.441e-01
Shapiro: 5.946e-31
```

```
P(t>1): 3.140e-01

# Match 3-Node Cycle
Matches: 1
Z-score: 0.970
P-value 1.661e-01
Shapiro: 3.195e-42
P(t>1): 7.100e-02

# Match Feed-Forvard Loop
Matches: 3
Z-score: 2.201
P-value 1.387e-02
Shapiro: 6.392e-34
P(t>3): 1.600e-02
```

Exercise 2: Anaysis of the regulation network from RegulonDB.

Download from the RegulonDB database *E. coli* K-12 Transcriptional Regulatory Network.

[ ]:
```
# Download the regulatory network
!wget http://regulondb.ccg.unam.mx/menu/download/datasets/files/network_tf_gene.
↪txt
```

Parse the tsv file reporting the transcription factor-gene interactions. Select the interactions with strong supporting data and build a graph with networkx and naswer the following questions:

1. What is the trascription factor that regulates more genes? How many genes are activated and repressed?
2. What is the gene that is regulated by more trascription factors? Which fraction of them are activating or repressing the expression of this gene?

Consider the Double-Positive Feedback loop

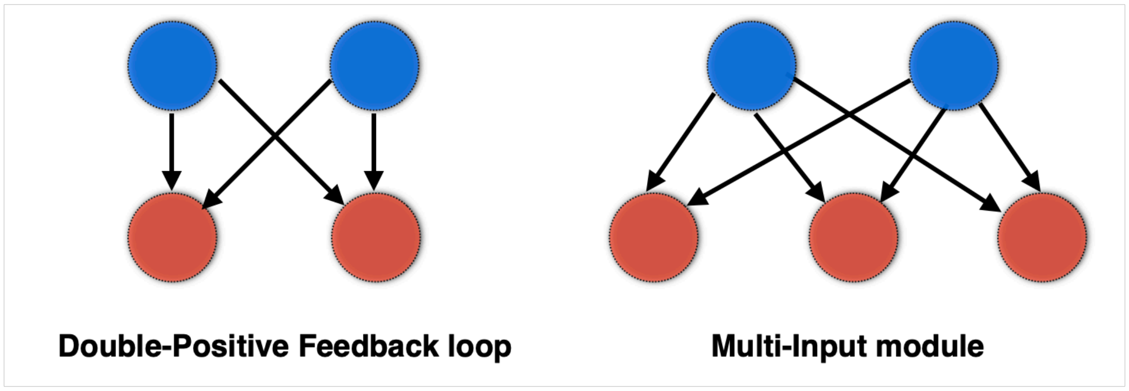
- N=[(1,'blue'),(2,'blue'),(3,'red'),(4,'red')]
- E=[(1,3,'+'),(1,4,'+'),(2,3,'+'),(2,4,'+')]

and Multi-Input module motifs

- N=[(1,'blue'),(2,'blue'),(3,'red'),(4,'red'),(5,'red')]
- E=[(1,3,'+'),(1,4,'+'),(1,5,'+'),(2,3,'+'),(2,4,'+'),(2,5,'+')]

[ ]:
```
from IPython.display import Image
Image('data/colored-motifs.png')
```

[ ]:



3. What is the number of motifs matched on the network?

[ ]:
```
#!/usr/bin/env python3
import sys
import networkx as nx
from networkx.algorithms import isomorphism
import matplotlib.pyplot as plt

# Function for parsing the tsv file

def read_net(filename):
        list_edges=[]
        f=open(filename,'r')
        for line in f:
                # Do not read header lines
                if line[0]=='#': continue
                line=line.rstrip('\n')
                v=line.split('\t')
                # Select data with  Strong supporting evidences
                if v[-1]!='Strong': continue
                # Define the sign of the edges
                if v[2]=='repressor':
                        sign='-'
                elif v[2]=='activator':
                        sign='+'
                else:
                        continue
                list_edges.append((v[0],v[1],sign))
        return list_edges


# Define the regulatory network assigning
# attributes to node and edges
```

```python
def get_net(list_edges):
        g=nx.MultiDiGraph()
        for e in list_edges:
                g.add_node(e[0],color='blue')
                g.add_node(e[1],color='red')
                g.add_edge(e[0],e[1],sign=e[2])
        return g


def motif_dpf():
        dpf=nx.MultiDiGraph()
        dpf.add_nodes_from([0,1],color='blue')
        dpf.add_nodes_from([2,3],color='red')
        dpf.add_edges_from([(0,2),(0,3),(1,2),(1,3)],sign='+')
        return dpf


def motif_mim():
        mim=nx.MultiDiGraph()
        mim.add_nodes_from([0,1],color='blue')
        mim.add_nodes_from([2,3,4],color='red')
        mim.add_edges_from([(0,2),(0,3),(0,4),(1,2),(1,3),(1,4)],sign='+')
        return mim


def show_graph(g):
        colors=[]
        for n in g.nodes:
                colors.append(g.nodes[n]['color'])
        nx.draw(g,node_color=colors)
        plt.show()


def get_max_degree(g,list_nodes):
        degs=[(g.degree(node),node) for node in list_nodes]
        degs.sort()
        degs.reverse()
        return degs[0][1],degs[0][0]


def get_elements(list_edges,pos):
        d={}
        for e in list_edges:
                d[e[pos]]=True
        return list(d.keys())
```

```python
def get_effect(g,node,effect,out=True):
        i=0
        if out:
                edges=g.out_edges(node)
        else:
                edges=g.in_edges(node)
        edges=list(set(edges))
        for e in edges:
                # The next line can change depending
                # on the version of networkx.
                se=g[e[0]][e[1]]
                for k in list(se.keys()):
                        if se[k]['sign']==effect: i=i+1
        return i


def match_isomorphisms(net,mot):
        diso={}
        em=isomorphism.categorical_multiedge_match('sign','+')
        nm=isomorphism.categorical_node_match('color','blue')
        ms=nx.isomorphism.DiGraphMatcher(net,mot,edge_match=em, node_match=nm)
        i=0
        for m in ms.subgraph_isomorphisms_iter():
                k=list(m.keys())
                k.sort()
                diso[tuple(k)]=True
                i=i+1
        i=len(list(diso.keys()))
        return i


if __name__ == '__main__':
        filename=sys.argv[1]
        list_edges=read_net(filename)
        list_tf=get_elements(list_edges,0)
        list_gene=get_elements(list_edges,1)
        g=get_net(list_edges)
        print ('Regulation Network')
        print (' Number of TFs: %d' %len(list_tf))
        print (' Number of Genes: %d' %len(list_gene))
        print (' Number of Edges: %d' %len(list_edges))
        max_tf=get_max_degree(g,list_tf)
        max_gene=get_max_degree(g,list_gene)
        act_tf=get_effect(g,max_tf[0],'+')
        rep_tf=get_effect(g,max_tf[0],'-')
        act_gene=get_effect(g,max_gene[0],'+',False)
        rep_gene=get_effect(g,max_gene[0],'-',False)
        print ('Max Degree TF: %s' %max_tf)
        print ('   Activation: %s' %act_tf)
        print ('   Repression: %s' %rep_tf)
        print ('Max Degree Gene: %s' %max_gene)
        print ('   Activation: %s' %act_gene)
        print ('   Repression: %s' %rep_gene)
        # Define the Double Positive Feedback Loop
        dpf=motif_dpf()
        n_dpf=match_isomorphisms(g,dpf)
        print ('Double Positive Feedback: %d' %n_dpf)
        # Define the Multi-Input Module
        mim=motif_mim()
        n_mim=match_isomorphisms(g,mim)
        print ('Multi-Input Module: %d' %n_mim)
        show_graph(g)
```

[4]:
```
# Run the scipt above giving in input the regulondb tsv file.
!python3 script/regulation_net.py data/network_tf_gene.txt
```

```
Regulation Network
 Number of TFs: 184
 Number of Genes: 1376
 Number of Edges: 2798
Max Degree TF: CRP
   Activation: 256
   Repression: 80
Max Degree Gene: csgG
   Activation: 7
   Repression: 5
Double Positive Feedback: 2141
Multi-Input Module: 8479
```

[ ]:
```
from IPython.display import Image
Image('data/regulon.png')
```

[ ]: