Dynamic networks Project:

> ### Analysis of trophic chains within a marine ecosystem

We considered a data set depicting carbon exchanges inside a marine ecosystem in Florida composed of 125 species and decomposition compartments. These species form complex food chains that involve carbon transfers and recycling. Food chains, also called **trophic chains**, consist in carbon transfers between different species that are often divided into 3 to 5 levels:

1. Decomposers
2. Primary producers that produce their food thanks to photosynthesis: algae, phytoplankton...
3. Primary consummers that eat primary producers: invertebrates such as corals, shrimps, lobsters...
4. Secondary consummers that eat primary consummers: fishes
5. Tertiary consumers that eat secondary ones: birds, reptiles, mammals...

etc.

Many ecosystems have **keystone species** that are important species in balancing the ecosystem and trophic chains.

This data consists in a directed graph indicating trophic chains in which the source node is eaten by the target node. We wanted to analyse this ecosystem and to identify keystone species in the ecosystem.

```
In [ ]: !pip install cdlib
```

In [3]:
```python
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import csv
from IPython.display import Image
from io import BytesIO
import scipy as sc
from scipy import stats
import os

from networkx import algorithms
from networkx.algorithms import centrality
from networkx.algorithms import assortativity

import cdlib
from cdlib import algorithms

'''To be replaced by local path of the user'''
os.chdir('C:\\Users\\lenovo\\Documents\\Cours\\M2\\Dynamic_Networks
```

I- Data treatment and obtaining of the graph

The data we used can be found at: http://networkrepository.com/eco-florida.php (http://networkrepository.com/eco-florida.php)

The original article that investigates the behaviour of the Florida marine ecosystem is in the folder attached to this notebook.

We looked for the names of the considered species in the original article and added labels to nodes.

The data set set is a .edges file indicating directed carbon transfers and their intensity. Nevertheless, we could not open this .edges file using networkx. To bypass this difficulty, we saved the text document listing the edges and their weight into a .csv file, before opening it in Python with the following code.

```
In [4]: def read_csv(name):
            with open(name, newline='') as csvfile:
                data = csv.reader(csvfile, delimiter=',', quotechar='|')
                array=[]
                k=0
                for row in data:
                    array.append([])
                    for x in row:
                        array[k].append(x)
                    k+=1
                array=np.array(array)
            return(array)


        Trees_edges=read_csv('eco-florida_edges.csv')
        Trees_nodes=read_csv('eco-florida_nodes.csv')
```

Then, from the obtained array we create a dictionary where keys are source nodes and values are a tuple containing the target node and the weigth of the carbon transfer.

```
In [5]: def array_to_dict(M):
            (n, k)=np.shape(M)
            D={}
            for i in range(1,n):
                Keys=list(D.keys())
                Node=int(M[i][0])
                if Node not in Keys:
                    D[Node]=[]
                    D[Node].append((int(M[i][1]), float(M[i][2])))
                else:
                    D[Node].append((int(M[i][1]), float(M[i][2])))
            return(D)

        Edgesdict=array_to_dict(Trees_edges)


        Nodesdict={}
        for i in Trees_nodes[1:]:
            Nodesdict[int(i[0])]=i[1]
```

Finally we built a graph from the edge dictionary that we get. To do so, the dictionary is converted into a list of 3-tuples containing source node, target node and the weight of carbon transfer

```
In [6]: def dictionary_to_edgelist(D):
            Keys=list(D.keys())
            Edges=[]
            for key in Keys:
                L=D[key]
                n=len(L)
                for i in range(n):
                    source=int(float(key))
                    target=int(float(D[key][i][0]))
                    weight=float(D[key][i][1])
                    Edges.append((source, target, weight))
            return(Edges)

        Edges=dictionary_to_edgelist(Edgesdict)
        #
        GFlorida=nx.DiGraph()
        #
        GFlorida.add_weighted_edges_from(Edges, weight='carbon_transfer')
        n,m=len(GFlorida.nodes()), len(GFlorida.edges())
        ER=nx.gnm_random_graph(n,m) #random Erdos Renyi graph we will use fo
```

Let's have a look at our data -

We vizualized the graph using Gephi, the raw graph and the graph with nodes colored by degree are attached to this notebook as "FloridaEcosystem_Raw.pdf" and "FloridaEcosystem_ColoredByDegree.pdf".

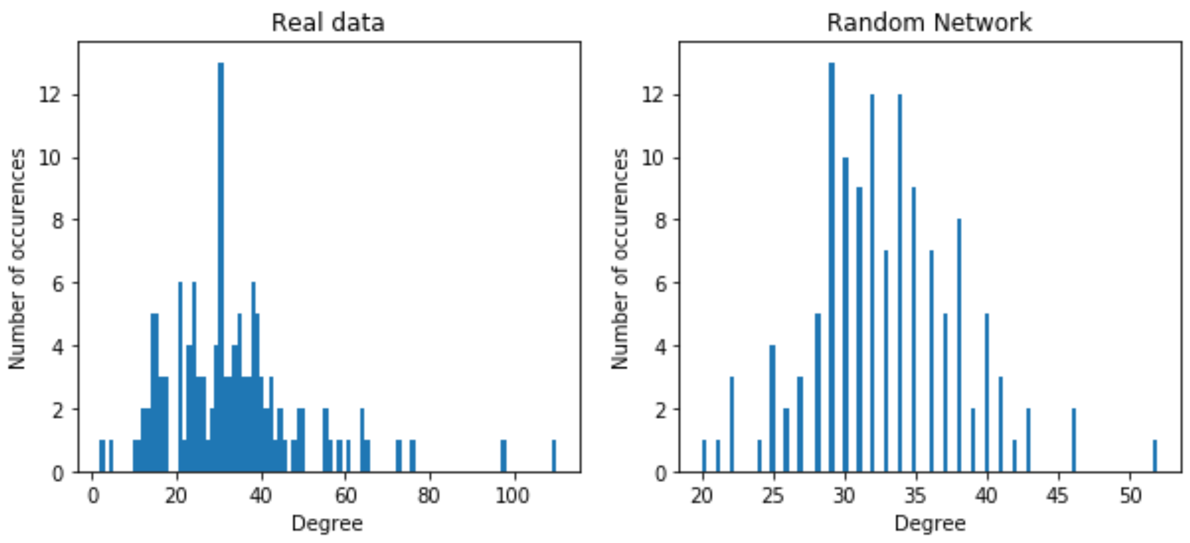
II- Data analysis: identification of keystone species

As a first approximation we wanted to test whether the structure of our real data can be recapitulated by a random graph.

To do so, we compared the distribution of degrees in the real graph and a random version of the graph using the Erdos-Renyi model with the same number of nodes and edges.

Plot Degree distribution

In [7]:
```python
#real data
Degreedict={}
for i in list(GFlorida.degree()):
        Degreedict[i[0]]=i[1]
Degreelist=[]
for i in range(1,n+1):
    Degreelist.append(Degreedict[i])
#comparative ER network
rdDegreedict={}
for i in list(ER.degree()):
        rdDegreedict[i[0]]=i[1]
rdDegreelist=list(rdDegreedict.values())

plt.figure(1, figsize=(10,4))

plt.subplot(1,2,1)
plt.hist(Degreelist, bins=100)
plt.xlabel('Degree')
plt.ylabel('Number of occurences')
plt.title('Real data')

plt.subplot(1,2,2)
plt.hist(rdDegreelist, bins=100)
plt.xlabel('Degree')
plt.ylabel('Number of occurences')
plt.title('Random Network')

plt.show()
```



We can see that the degree distribution of our data is way broader than the degree distribution of a random Erdos renyi graph built with the same number of nodes and edges. Our data set cannot be recapitulated by a random graph.

Cummulated carbon flow for each nodes

We call flow the carbon flow between two nodes (species or detritic compartment). We hypothesized that the keystone species of the ecosystem are the species that exchange a lot of carbon and that have a prominent position in the ecosystem.

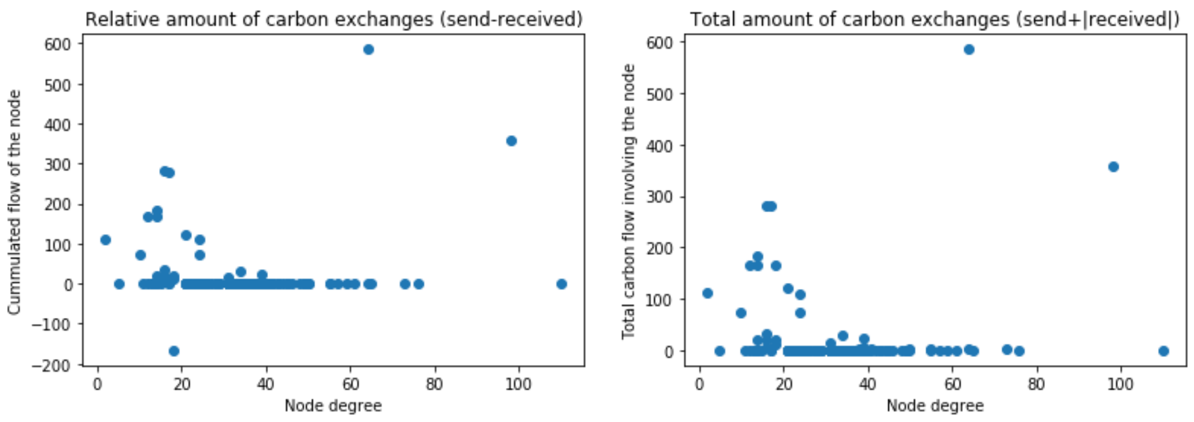
Thus we first compute the cummulated carbon exchange for each node. We also tried to link the importance of carbon exchange by the number of links that a specie or a compartment has with other species/compartments.

In [8]:
```
#List of cummulated carbon flux for each node
CummulFlow=[0 for i in range(n)]
for i in range(1,len(Edgesdict)):
    CummulFlow[i-1]=sum(k[1] for k in Edgesdict[i]) #add positive f
    for j in Edges: #remove negative flows
        if j[1]==i:
            CummulFlow[i]-= j[2]

#List of the total amount of carbon a node exchanges with its neigh
TotFlow=[0 for i in range(n)]
for i in range(1,len(Edgesdict)):
    TotFlow[i-1]=sum(k[1] for k in Edgesdict[i]) #add positive flow
    for j in Edges: #remove negative flows
        if j[1]==i:
            TotFlow[i]+= j[2]

#We add the cummulative flow as a node attribute
Cummul_flow={}
for node in list(GFlorida.nodes()):
  Cummul_flow[node]=CummulFlow[node-1]
nx.set_node_attributes(GFlorida, Cummul_flow, 'cummul_flow')


#Identification of key carbon exchangers
def get_highcarbon_species(cummulflowdict,p):
  sortedbtw=sorted(cummulflowdict.items(), key=lambda item: item[1]
  highest_cflow=[]
  for i in range(1,p+1):
    highest_cflow.append(Nodesdict[sortedbtw[-i][0]])
  return highest_cflow
```

In [9]:
```
get_highcarbon_species(Cummul_flow,5)
```

Out[9]:
```
['Benthic POC',
 'Water POC',
 'Water flagellates',
 'Thalassia',
 'Benthic flagellates']
```

In [10]:
```python
#Plot carbon flux according to node degree
Degreepernode=[0 for i in range(n)]
for i in range(1,n+1):
    Degreepernode[i-1]=Degreedict[i]

plt.figure(2, figsize=(13,4))
plt.subplot(1,2,1)
plt.scatter(Degreepernode,CummulFlow)
plt.xlabel('Node degree')
plt.ylabel('Cummulated flow of the node')
plt.title('Relative amount of carbon exchanges (send-received)')

plt.subplot(1,2,2)
plt.scatter(Degreepernode,TotFlow)
plt.xlabel('Node degree')
plt.ylabel('Total carbon flow involving the node')
plt.title('Total amount of carbon exchanges (send+|received|)')
plt.show()
```



We observed that the highest carbon exchanges involve the detritic compartments (Benthic POC and Water POC) and some primary producers (flagellates that are unicellular organisms and thalassia that is a type of algae).

Nevertheless, there is no correlation between the degree of the node and the amount of carbon it exchanges. The total amount of carbon exchanged is not directly linked to the number of exchanges between the considered node and its neighbors.

Betweenness centrality

Another important parameter to characterize keystone species in an ecosystem is their place is the network of carbon exchanges. If we remove a keystone specie from the ecosystem, trophic chains would be imbalances.

Then we used the betweenness centrality to identify determinant species in the carbons exchanges.

To better analyze the relationship between communities and node degree, betweenness or carbon flux we also plotted the mean of the considered centrality in each community.

In [11]:
```python
#This functions return a dictionary whose keys are species indexes a
def get_betweeness(G, w):
    dict_betweeness=nx.algorithms.centrality.betweenness_centrality
    return(dict_betweeness)


#This function returns the names of the p species whith the highest
def get_imp_species(betweenessdict,p):
  sortedbtw=sorted(betweenessdict.items(), key=lambda item: item[1]
  highest_betweeness=[]
  for i in range(1,p+1):
    highest_betweeness.append(Nodesdict[sortedbtw[-i][0]])
  return highest_betweeness

get_imp_species(get_betweeness(GFlorida, 'carbon_transfer'),5)
```

Out[11]:
```
['Water POC', 'Other demersals fishes', 'Meroplankton', 'Flatfishe
s', 'Loon']
```

We can observe that the two nodes with the highest betweeness correspond to detritic compartments (Benthic POC and Water POC). The most important species are small invertebrates that are primary consummers.

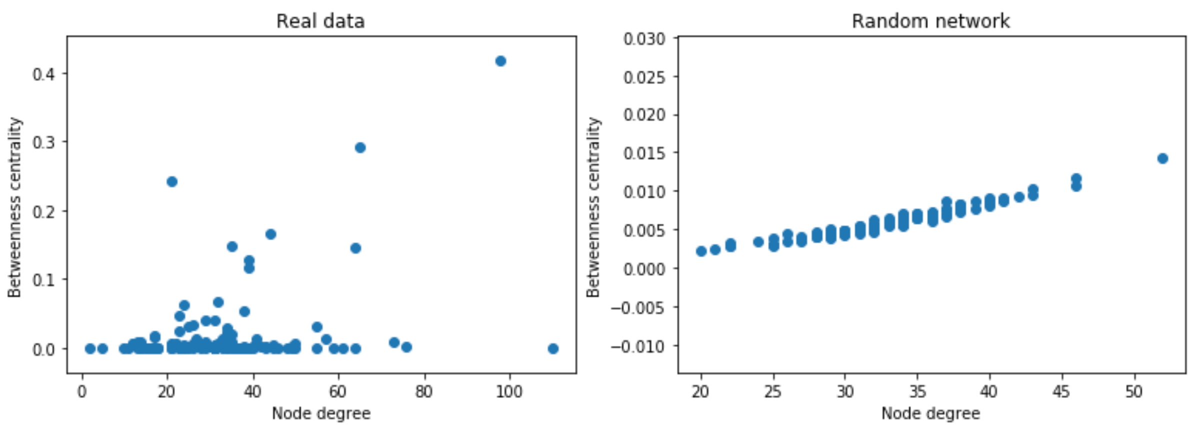
Interestingly, our algorithm allows the detection of detritical compartments that are the nodes with the highest betweenness and carbon echanges. Species of highest betweeness are primary consummers while species with high carbon exchanges are primary producers.

This is compatible with intuition. On the one hand, detritical compartments and primary producers receive a lot of carbon from dead animals and plants, and are at the base of trophic chains so are a source of food for a lot of other species. On the other hand, primary consummers have a higher betweenness because they are low on the trophic chain to they are a link between many higher consummers and the producers of food.

Then we wondered whether the betweenness of nodes depends on their degree. Are nodes of high betweenness,which serve as bridges between different parts of the network, also nodes of high degree? To answer this question we analyzed the homophyly between node degree and betweenness centrality.

As betweenness centrality does not take into account the directionnality of graphs, we can compare the results obtain with real dat with the results obtained with a random undirected graph having the same number of edges and nodes.

In [12]:
```python
def plot_betweenness(G, w=None, thresh=None, pos=None):
    Degree=[]
    Betweenness=[]
    L=list(G.degree())
    dict_betweenness=get_betweeness(G, w)
    Keys=list(dict_betweenness.keys())
    n=len(Keys)
    if pos=='sup':
        for i in range(n):
            if dict_betweenness[Keys[i]]>thresh:
                Degree.append(L[i][1])
                Betweenness.append(dict_betweenness[Keys[i]])
    elif pos=='inf':
        for i in range(n):
            if dict_betweenness[Keys[i]]<thresh:
                Degree.append(L[i][1])
                Betweenness.append(dict_betweenness[Keys[i]])
    else:
        for i in range(n):
            Degree.append(L[i][1])
            Betweenness.append(dict_betweenness[Keys[i]])
    return [Degree, Betweenness]


plt.figure(3, figsize=(13,4))
plt.subplot(1,2,1)
Degree=plot_betweenness(GFlorida,'carbon_transfer')[0]
Btw=plot_betweenness(GFlorida,'carbon_transfer')[1]
plt.scatter(Degree, Btw)
plt.xlabel('Node degree')
plt.ylabel('Betweenness centrality')
plt.title('Real data')

plt.subplot(1,2,2)
Degreerd=plot_betweenness(ER)[0]
Btwrd=plot_betweenness(ER)[1]
plt.scatter(Degreerd, Btwrd)
plt.xlabel('Node degree')
plt.ylabel('Betweenness centrality')
plt.title('Random network')

plt.show()

print('Pearson corelation coefficient for real data:',stats.pearson
```



```
Pearson corelation coefficient for real data: 0.346633452465892 pv
alue: 6.117352519452672e-05
```

We can observe that there is a homophily between node degree and betweenness, as indicated by a positive Pearson correlation coefficient. Nevertheless, this homophily is much smaller than the one observed for the random graph. This positive correlation between node degree and betweenness could be due to the detritic compartments that have both high degree and high betweeness. For species, species that are linked to many other are not particularly the most important ones, as defined by betweenness.

III- Data analysis: Study of communities

Community analyses

This section aims at finding and understanding the different communities that may be present within the ecosystem. We wondered whether species within this ecosystem are best grouped according to the trophic chain they belong to, or according to their position in the trophic chain. For instance, do communities correspond to all primary producers, all secondary producers etc., or do they correspond to the different trophic chains in the ecosystems from primary producers to secondary predators?

> 1) Identification of communities and link with important node parameters

Community analysis using the Louvain algorithm can only be performed on undirected graphs.

First, we separated the communities in a undirected version of the graph using the Louvain algorithm.

In [13]:
```
Und_G=GFlorida.to_undirected()
Communities=algorithms.louvain(Und_G)
Dict_commu=Communities.to_node_community_map()
indexmax_commu=(max(Communities.to_node_community_map().values()))
nb_commu=indexmax_commu[0]+1
print("There are", nb_commu, "communities in this ecosystem" )
```

```
There are 4 communities in this ecosystem
```

```
In [39]: '''takes a dictionary that contains nodes as keys and the list of c
         to as values, and returns a dictionary with communities as keys and
         values'''
         def get_community_nodes(Dict_comm):
             Dict_nodes={}
             Communities=[]
             nodes=list(Dict_comm.keys())
             n=len(nodes)
             for i in range(1,n+1):
                 community=Dict_comm[i][0]
                 if community not in Communities:
                     Dict_nodes[community]=[i]
                     Communities=list(Dict_nodes.keys())

                 else:
                     Dict_nodes[community].append(i)
             return (Dict_nodes)
         '''program to get the modularity of the program on a partition give
         in communities'''

         def get_modularity(G):
             Communities=cdlib.algorithms.louvain(G)
             Dict_communities=Communities.to_node_community_map()
             Dict_comm=dict(Dict_communities)
             Dict_nodes=get_community_nodes(Dict_comm)
             Keys=list(Dict_nodes.keys())
             Partition=[]
             for key in Keys:
                 Partition.append(Dict_nodes[key])
             mod=nx.algorithms.community.modularity(G, Partition)
             return(mod)
```

```
In [40]: get_modularity(Und_G)
```

```
Out[40]: 0.1806246479895475
```

```
In [34]: Communities=cdlib.algorithms.louvain(Und_G)
         Dict_communities=Communities.to_node_community_map()
         Dict_comm=dict(Dict_communities)
         get_community_nodes2(Dict_comm)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent c
all last)
<ipython-input-34-2aa34b305cfe> in <module>
      2 Dict_communities=Communities.to_node_community_map()
      3 Dict_comm=dict(Dict_communities)
----> 4 get_community_nodes2(Dict_comm)

<ipython-input-30-7144f7b9821c> in get_community_nodes2(Dict_comm)
     22     n=len(nodes)
     23     for i in range(n):
---> 24         community=Dict_comm[i][0]
     25         print(community)
     26         if community not in Communities:

KeyError: 0
```

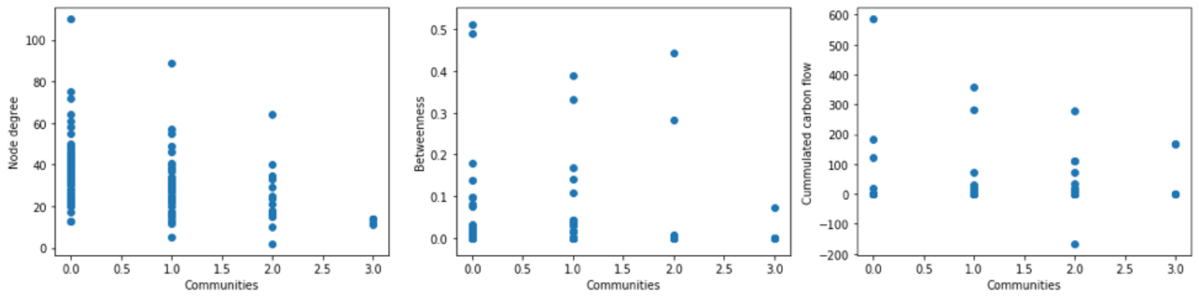
There are 4 communities in the ecosystem.

We evaluated the quality of the partition using modularity score.The value of 0.18 shows that the connectivity within the communities is higher than within a random graph, but that the intra-community connectivity remains low. This is in line with the hypothesis that they correspond to the different levels of trophic chains such as soil & decomposers, primary producers such as algae, primary consummers, and secondary consummers such as birds. Under this hypothesis, the internal connection within each community would not be very high as each community capture species at the same trophic level but belonging to different trophic chains.

To test the plausibility of this hypothesis, we tried to characterize the identified communities using node characteristics such as betweenness or degree. We plotted the distribution of node degree, betweenness centrality and average carbon flows across communities

```
In [41]: #takes a dictionary that has lists of one element in front of the ke
         def convert_values(D):
             new_D={}
             Keys=list(D.keys())
             n=len(Keys)
             for i in range(n):
                 new_D[Keys[i]]=D[Keys[i]][0]
             return(new_D)

         #plots the communities in ordinates as a function of another parame
         def plot_communities(G, abscissa, weight):
             Communities=algorithms.louvain(G, weight)
             Dict_communities=convert_values(Communities.to_node_community_ma
             X_values=[]
             comm_plot=[]
             Keys=list(Dict_communities.keys())
             n=len(Keys)
             if abscissa=='degree':
                 for i in range(n):
                     X_values.append(G.degree(Keys[i]))
                     comm_plot.append(Dict_communities[Keys[i]])
             if abscissa=='cummul_flow':
                 for i in range(n):
                     X_values.append(G.nodes[Keys[i]]['cummul_flow'])
                     comm_plot.append(Dict_communities[Keys[i]])
             if abscissa=='betweenness':
                 dict_betweeness=nx.algorithms.centrality.betweenness_centra
                 for i in range(n):
                     X_values.append(dict_betweeness[Keys[i]])
                     comm_plot.append(Dict_communities[Keys[i]])
             return[X_values,comm_plot]
```

In [42]:
```
plt.figure(5, figsize=(18,4))
plt.subplot(1,3,1)
Y=plot_communities(Und_G, abscissa='degree',weight='carbon_transfer
X=plot_communities(Und_G, abscissa='degree',weight='carbon_transfer
plt.scatter(X, Y)
plt.xlabel('Communities')
plt.ylabel('Node degree')

plt.subplot(1,3,2)
Y=plot_communities(Und_G, abscissa='betweenness',weight='carbon_tra
X=plot_communities(Und_G, abscissa='betweenness',weight='carbon_tra
plt.scatter(X, Y)
plt.xlabel('Communities')
plt.ylabel('Betweenness')

plt.subplot(1,3,3)
Y=plot_communities(Und_G, abscissa='cummul_flow',weight='carbon_tra
X=plot_communities(Und_G, abscissa='cummul_flow',weight='carbon_tra
plt.scatter(X, Y)
plt.xlabel('Communities')
plt.ylabel('Cummulated carbon flow')

plt.show()
```

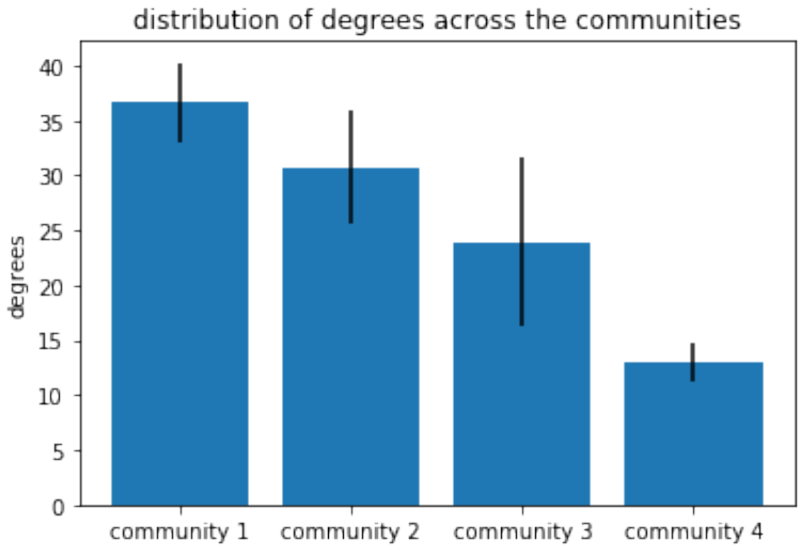


To better analyze the relationship between communities and node degree, betweenness or carbon flux we also plotted the mean of the considered centrality in each community.

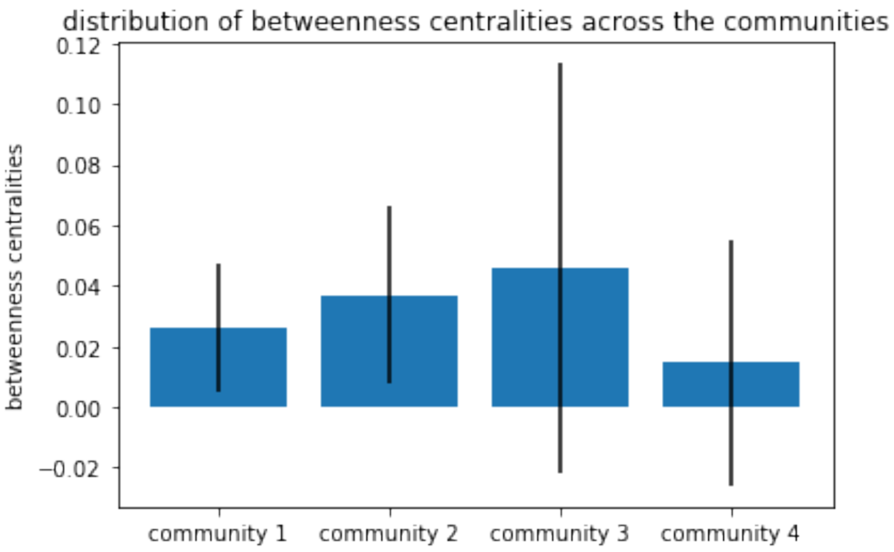
In [43]:
```
# get the confidence intervals around the different values

def mean_confidence_interval(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), sc.stats.sem(a)
    h = se * sc.stats.t.ppf((1 + confidence) / 2., n-1)
    return h

#gets a dictionary with the different communities as keys and the l
#in those different communities as entries, properties can be degre

def get_community_properties(G, weight, prop):
    Communities=algorithms.louvain(G,weight)
    Dict_communities=convert_values(Communities.to_node_community_ma
    Keys=list(Dict_communities.keys())
    n=len(Keys)
    Property={0:[], 1:[], 2:[], 3:[]}
    if prop=='betweenness':
```

```
    if prop=='betweenness':
      dict_betweenness=nx.algorithms.centrality.betweenness_central
      for i in range(n):
          for community in list(Property.keys()):
              if Dict_communities[Keys[i]]==community:
                  Property[community].append(dict_betweenness[Keys[
    if prop=='cummul_flow':
        for i in range(n):
            for community in list(Property.keys()):
                if Dict_communities[Keys[i]]==community:
                    Property[community].append(G.nodes[Keys[i]]['cu
    if prop=='degree':
        for i in range(n):
            for community in list(Property.keys()):
                if Dict_communities[Keys[i]]==community:
                    Property[community].append(G.degree[Keys[i]])
    return(Property)


#plots the means of the tested properties across the communities, c
#Also implements the error bars of the 95% confidence interval (ass

def plot_mean_props(G, weight, prop, ylabel):
    fig, ax=plt.subplots()
    Property=get_community_properties(G, weight, prop)
    Communities=list(Property.keys())
    n=len(Communities)
    Labels=['community '+str(Communities[i]+1) for i in range(n)]
    x_pos=np.arange(len(Labels))
    List_bars=[]
    List_errors=[]
    for community in Communities:
        List_bars.append(np.mean(Property[community]))
        List_errors.append(mean_confidence_interval(Property[commun
    ax.bar(x_pos, List_bars, yerr=List_errors)
    ax.set_xticks(x_pos)
    ax.set_xticklabels(Labels)
    plt.ylabel(ylabel)
    plt.title('distribution of '+ylabel+' across the communities', 
    plt.show()
```

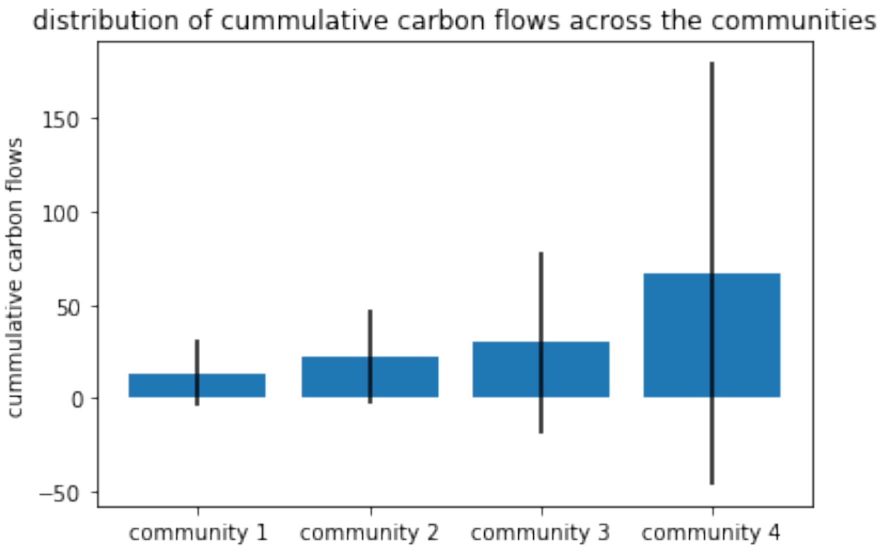
```
In [44]: plot_mean_props(Und_G,weight='carbon_transfer', prop='degree', ylab
```



```
In [45]: plot_mean_props(Und_G, weight='carbon_transfer', prop='betweenness'
```



```
In [46]: plot_mean_props(Und_G, weight='carbon_transfer', prop='cummul_flow'
```



```
In [47]: #Number of elements inside each community for the GFlora ecosystem
         def nb_in_commu(G,weight):
           Communities=algorithms.louvain(G,weight)
           Dict_communities=Communities.to_node_community_map()
           nb_commu=max(Dict_communities.values())[0]+1
           C=[[] for i in range(nb_commu)]
           Nodes=list(G.nodes())
           for node in Nodes:
             j=Dict_commu[node][0]
             C[j].append(node)
           Imp_commu=[len(C[i]) for i in range(nb_commu)]
           return(Imp_commu)
```

```
In [53]: nb_in_commu(Und_G,'carbon_transfer')
```

```
Out[53]: [70, 37, 16, 5]
```

Those results allow to characterize the different communities present in the ecosystem. Nevertheless we noted that the number of species/compartments in each community is very imbalanced. This makes difficult the comparison of means. Moreover, errors bars are huge here and none of the betweenness centralities of carbon flow are statistically different between communities.

We can make qualitative comparisons betweenn the obtained communities to get insights into what they could represent.

- Community 1: high BC, highest degree, low carbon flow:

This may correspond to detritical compartments and primary producers. They are at the base of trophic chains: they receive carbon from dead beings but they undergo high carbon uptake, which is consistent with low cummulated carbon flow and high degree.

- Community 2: similar profile with lower degrees but higher BC and average carbon flows:

This may correspond to primary consummers, as we saw earlier that they have the highest betweenness centrality. They have quite a central role (high degrees), but relatively low carbon flows could indicate that there is predation/carbon uptake, which seems consistent with primary consummers.

- Community 3: low carbon flows and low degrees:

BC centrality is hard to analyze here. This could correspond to primary or secondary consummers. They are at intermediate levels of trophic chains which could explain the low carbon flow (their eat as much as they get eaten by other species). But they are high enough in trophic chains to have few links to other species.

- Community 4: the highest carbon flows, with the lowest degrees and BC:

This could correspond to the highest-level of consummers or predators. They don't have a lot of predators so their intake of carbon is positive: they withdraw more food from the network than they serve as food. They are at the top of the trophic chains which could account for their low degree and betweenness centrality.

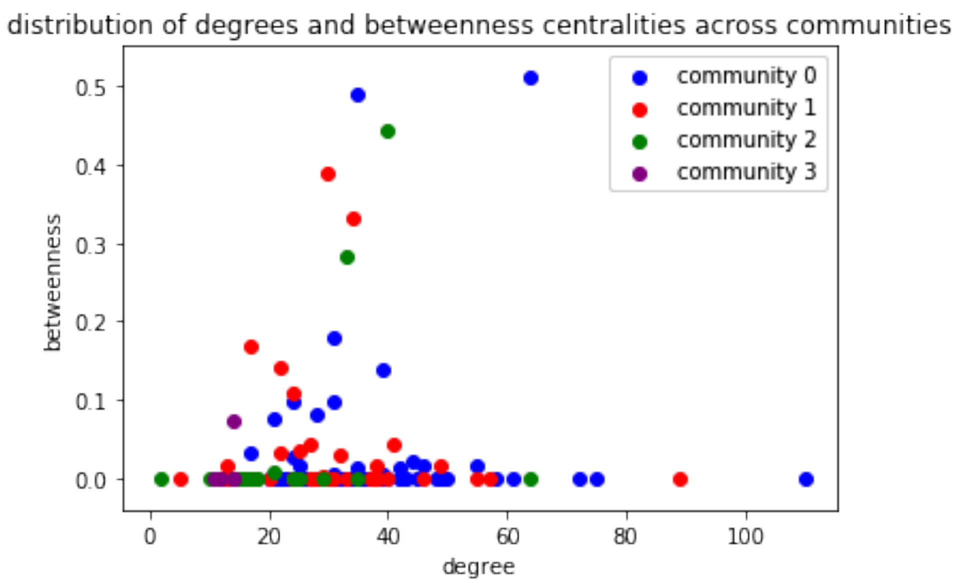
We still need to compare these hypotheses with the nodes label. Moreover, the huge error bars make the interpretation difficult.

We also plotted the different communities with two of their characteristics: degree (x-axis) and average carbon flows or betweenness (y-axis)

In [49]:
```
#gets 2 dictionaries, the keys are the different communities, the e
#for X, it's degrees, for Y, it can be betweenness or average flow

def distinguish_communities(G, weight, prop):
    Communities=algorithms.louvain(G,weight)
    Dict_communities=convert_values(Communities.to_node_community_m
    Keys=list(Dict_communities.keys())
    n=len(Keys)
    X_value={0:[], 1:[], 2:[], 3:[]}
    Y_value={0:[], 1:[], 2:[], 3:[]}
    if prop=='betweenness':
        dict_betweeness=nx.algorithms.centrality.betweenness_centra
        for i in range(n):
            for community in list(X_value.keys()):
                if Dict_communities[Keys[i]]==community:
                    X_value[community].append(G.degree[Keys[i]])
                    Y_value[community].append(dict_betweeness[Keys[
    if prop=='cummul_flow':
        for i in range(n):
            for community in list(X_value.keys()):
                if Dict_communities[Keys[i]]==community:
                    X_value[community].append(G.degree[Keys[i]])
                    Y_value[community].append(G.nodes[Keys[i]]['cumm
    return(X_value, Y_value)


#plots the nodes with degree in abscissa and average flow and betwe
#different colours are used for the different communities, the lege

def plot_gap_communities_with_legend(G, weight, prop):
    fig, ax=plt.subplots()
    X_value, Y_value=distinguish_communities(G, weight, prop)
    Colours=['blue', 'red', 'green', 'purple']
    Communities=list(X_value.keys())
    for i in range(4):
        ax.scatter(X_value[Communities[i]], Y_value[Communities[i]]
    plt.xlabel('degree')
    plt.ylabel(prop)
    legend = ax.legend()
    ax.add_artist(legend)
    if prop=='betweenness':
        t_prop='betweenness centralities'
    if prop=='cummul_flow':
        t_prop='cummulative carbon flows'
    plt.title('distribution of degrees and '+t_prop+' across commun
    plt.show()
```

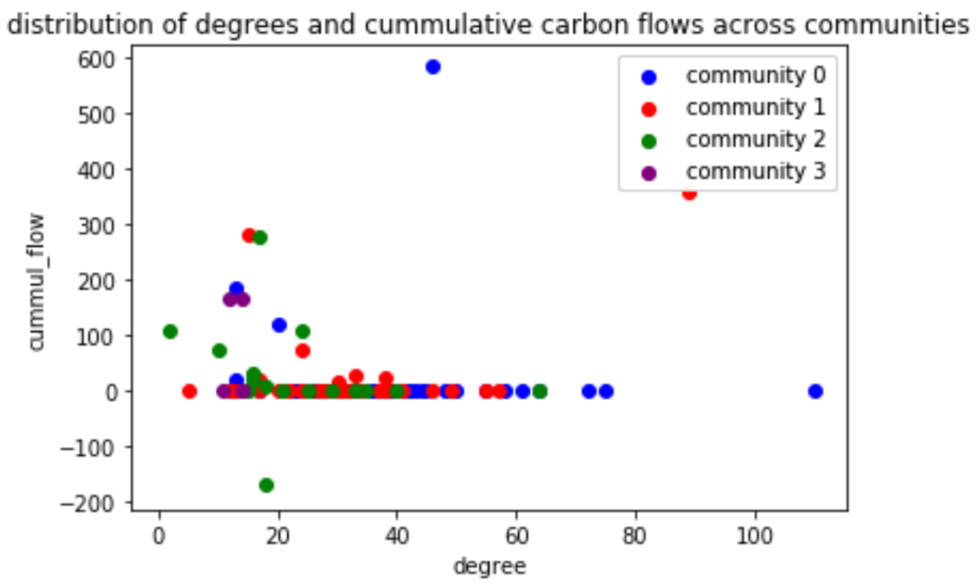
Plot with betweenness and degrees

```
In [50]: plot_gap_communities_with_legend(Und_G, weight='carbon_transfer', p
```



Plot with cummulated flow and degree

```
In [51]: plot_gap_communities_with_legend(Und_G, weight='carbon_transfer',pr
```



No particular tendency can be observed from these plots. No combination of parameters can be used to efficiently distinguish communities.

Comparison of the communities with theoretical communities consisting in the different trophic levels

Finally, we wanted to test whether the comunity partition given by the Louvain algorithm correspong to the different levels of trophic chains by directly looking at the species inside communities.

To do so, we converted the obtained communities into 4 lists containing the name of species within the same community.

In [52]:
```
Communities=algorithms.louvain(Und_G, 'carbon_transfer')
Dict_commu=Communities.to_node_community_map()
nb_commu=max(Dict_commu.values())[0]+1
L_commu=[[] for i in range(nb_commu)]
for node in list(Und_G.nodes()):
  community=Dict_commu[node][0]
  L_commu[community].append(Nodesdict[node]) #add the name of the c

for i in range(nb_commu):
  print('Community',i+1,':', len(L_commu[i]),'nodes', L_commu[i])
```

```
Community 1 : 70 nodes ['Stone crab', 'Herbivorous amphipods', 'Pr
edatorous shrimp', 'Pink shrimp', 'Omnivorous crabs', 'Pinfish', '
Other demersals fishes', 'Herbivorous ducks', 'Omnivorous ducks',
'Benthic POC', 'Unknown3', 'Sharks', 'Rays', 'Bonefish', 'Sardines
', 'Catfish', 'Eels', 'Toadfish', 'Brotulas and Batfishes', 'Golds
potted killifish', 'Snooks', 'Seahorses', 'Gulf pipefish', 'Dwarf
seahorse', 'Snappers', 'Gray snapper', 'Mojarras', 'Porgies', 'Sci
aenids', 'Spadefish', 'Blennies', 'Flatfishes', 'Loon', 'Greeb', '
Pelican', 'Comorant', 'Big herons and egrets', 'Small herons and e
grets', 'Ibis', 'Roseate Spoonbill', 'Predatory ducks', 'Fruiforme
s', 'Small shorebirds', 'Gulls and terns', 'Kingfisher', 'Dolphin'
, 'Spiny lobster', 'Tarpon', 'Pufferfishes', 'Callinectes species'
, 'Filefishes', 'Raptors', 'Barracudas', 'Predatory gasteropods',
'Other cnidaria', 'Echinodermata', 'Code Goby', 'Clown Goby', 'Ben
thic flagellates', 'Benthic cilliates', 'Meiofauna', 'Detritivorou
s gasteropods', 'Detritivorous polychaetes', 'Predatory polychaete
s', 'Macrobenthos', 'Benthic crustaceans', 'Detritivorous amphipod
s', 'Detritivorous crabs', 'Predatory crabs', 'Epiphyte grazing ga
steropods']
Community 2 : 37 nodes ['2um spherical phytoplancton', 'Synedococc
us', 'Small Diatoms', 'Meroplankton', 'Water POC', 'Green turtle',
'Mollies', 'Parrotfishes', 'Mullets', 'Anchovies', 'Bay anchovy',
'Lizardfish', 'Silversides', 'Jacks and runners', 'Pompano and per
mits', 'Grunts', 'Spotted seatrout', 'Red drum', 'Other pelagics f
ishes', 'Crocodiles', 'Loggerhead', 'Hawksbill turtle', 'Needlefis
hes', 'Mackerels', 'Water flagellates', 'Water ciliates', 'Acartia
tonsa', 'Oithona nana', 'Paracalanus', 'Other Copepoda', 'Other zo
oplankton', 'Sponges', 'Bivalves', 'Pelagic feeding polychaetes',
'Coral', 'Killifishes', 'Rainwater killifish']
Community 3 : 16 nodes ['Unknown1', 'Big Diatoms', 'Benthic microa
lgae', 'Thalassia', 'Halodule', 'Syringodium', 'Roots', 'Drift alg
ae', 'Epiphytes', 'Unknown2', 'Isopods', 'Halfbeaks', 'Manatee', '
Groupers', 'Herbivorous shrimp', 'Grass shrimp']
Community 4 : 5 nodes ['Oscillatoria', 'Dinoflagellates', 'Other p
hytoplankton', 'DOC', 'Free bacteria']
```

Here we can observe that the detritic compartments 'DOC', 'Water POC' and 'Benthic DOC' are in separated communities (respectively in the communities 4, 2 and 1). Another exemple is that the community 2 gathers predators such as crocodiles and primary consummers such as meroplanktons. These seem to disagree with our hypothesis that the 4 communities separate species according to their level in trophic chains.

Nevertheless, we can observe that the first community contains many fishes and birds that are secondary and tertiary consummers. The second community contains a lot of small fishes such as anchoives, killifishes and other primary consummers such as cilliates. The third community is mainly composed of primary producers such as epiphytes (they are plants), algae, micro algae or diatoms. The fourth community is composed of small invertebrates that are also primary producers.

To conclude there a maybe tendencies in communities that group species with similar levels in trophic chains. But this partition clearly does not separate species according to their trophic level.

Other parameters influence modularity, then partition into communities.

IV- Assortativity analysis

Assortativity measurements allow to understand whether a specific set of nodes builds edges with neighbors that share the same properties. Here, we propose to compute the assortativity regarding the different properties in the four communities. Finding a higher assortativity regarding a specific parameter in the communities compared to the basic graph would suggest that the clustering in communities grouped nodes based on that property.

On the other hand, finding disassortativity within the communities would indicate that the clustering tended to group nodes of different properties, which would also provide valuable informations over the relationships captured within the communities.

All in all, we expect that analyzing the assortativity in our different communities regarding betweenness, degree and average carbon flow between species will help to understand what groups species within the same community, and thus to get a better idea on the structuration of the ecosystem.

We computed the assortativity of an undirected graph regarding the different communities.

In [54]:
```
def get_community_assortativity(G, weight):
    Communities=algorithms.louvain(G, weight)
    Dict_communities=Communities.to_node_community_map()
    nx.set_node_attributes(G, Dict_communities, 'Community')
    community_assortativity=nx.algorithms.assortativity.attribute_a
    return(community_assortativity)
```

The next functions allow to run distinct analyses on the assortativity of different communities.

In [55]:
```
'''takes a dictionary that contains nodes as keys and the list of c
to as entries, and returns a dictionary with communities as keys an
entries'''

def get_community_nodes(Dict_comm):
    Dict_nodes={}
    Communities=list(Dict_nodes.keys())
    Keys=list(Dict_comm.keys())
    n=len(Keys)
    for i in range(n):
        community=Dict_comm[Keys[i]]
        if community not in Communities:
            Dict_nodes[community]=[Keys[i]]
            Communities=list(Dict_nodes.keys())
        else:
            Dict_nodes[community].append(Keys[i])
    return(Dict_nodes)


'''takes a graph, a property and a list of nodes, and returns a dic
nodes as keys and their property as entry, can work with betweennes

def node_properties(G, list_nodes, prop):
    Dict_prop={}
    n=len(list_nodes)
    if prop=='betweenness':
        dict_betweeness=nx.algorithms.centrality.betweenness_centra
        for i in range(n):
            Dict_prop[list_nodes[i]]=dict_betweeness[list_nodes[i]]
    else:
        for i in range(n):
            Dict_prop[list_nodes[i]]=G.nodes[list_nodes[i]][prop]
    return(Dict_prop)
```

Then, we separated the subgraphs generated by the different communities in order to get the differences in assortativity between the communities.

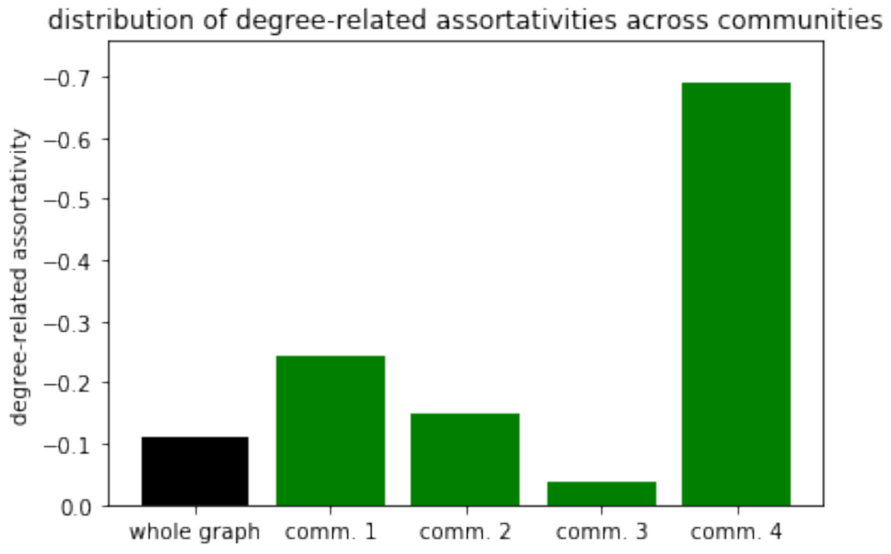
In [56]:
```
'''gets a dictionary that contains communities as keys and associat
as entries, will be useful to get assortativity within communities'

def get_community_graphs(G):
    Dict_subgraphs={}
    Communities=cdlib.algorithms.louvain(G)
    Dict_comm=convert_values(Communities.to_node_community_map())
    Dict_nodes=get_community_nodes(Dict_comm)
    Comm_list=list(Dict_nodes.keys())
    n=len(Comm_list)
    for i in range(n):
        Sub_nodes=Dict_nodes[Comm_list[i]]
        Sub_graph=G.subgraph(Sub_nodes)
        Dict_flows=node_properties(G, Sub_nodes, 'cummul_flow')
        nx.set_node_attributes(Sub_graph, Dict_flows, 'cummul_flow'
        Dict_betweenness=node_properties(G, Sub_nodes, 'betweenness
        nx.set_node_attributes(Sub_graph, Dict_betweenness, 'betwee
        Dict_subgraphs[Comm_list[i]]=Sub_graph
    return(Dict_subgraphs)
```

Finally, we obtained and plotted the assortativities associated to different properties in the different communities. A high assortativity will indicate some homogeneity in the species linked in the different communities.

```
In [57]: '''takes a graph and a property (degree, betweenness or average_flo
         of communities and returns the assortativity associated to all the 
         on the targeted property'''

         def get_crosscomm_assortativity(G, prop):
             List_assortativity=[]
             Dict_subgraphs=get_community_graphs(G)
             Communities=list(Dict_subgraphs.keys())
             n=len(Communities)
             if prop=='degree':
                 degree_assortativity=nx.algorithms.assortativity.degree_ass
                 List_assortativity.append(degree_assortativity)
                 for i in range(n):
                     comm_graph=Dict_subgraphs[Communities[i]]
                     degree_assortativity=nx.algorithms.assortativity.degree
                     List_assortativity.append(degree_assortativity)
             else:
                 prop_assortativity=nx.algorithms.assortativity.attribute_as
                 List_assortativity.append(prop_assortativity)
                 for i in range(n):
                     comm_graph=Dict_subgraphs[Communities[i]]
                     prop_assortativity=nx.algorithms.assortativity.attribut
                     List_assortativity.append(prop_assortativity)
             return(List_assortativity)


         '''plots the assortativity of the different communities of the grap
         property given as an input'''

         def plot_assortativities(G, prop):
             Colors=['black']
             Communities=['whole graph']
             for i in range(1, 5):
                 Communities.append('comm. '+str(i))
                 Colors.append('green')
             List_assortativity=get_crosscomm_assortativity(G, prop)
             plt.ylim(0, 1.1*max(List_assortativity, key=abs))
             plt.bar(Communities, List_assortativity, color=Colors)
             plt.ylabel(prop+'-related assortativity')
             plt.title('distribution of '+prop+'-related assortativities acr
             plt.show()
```

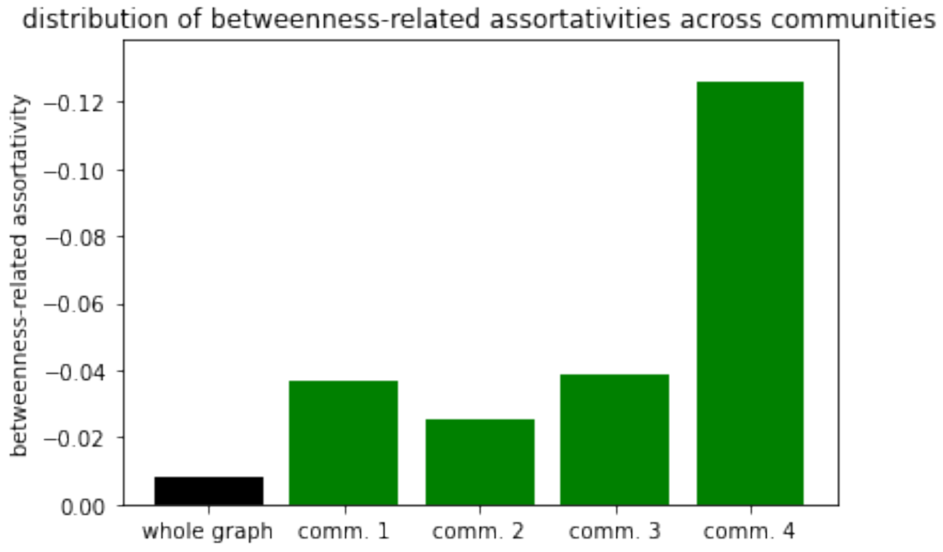
Degree-related assortativities in all the different communities

```
In [59]: plot_assortativities(Und_G, 'degree')
```



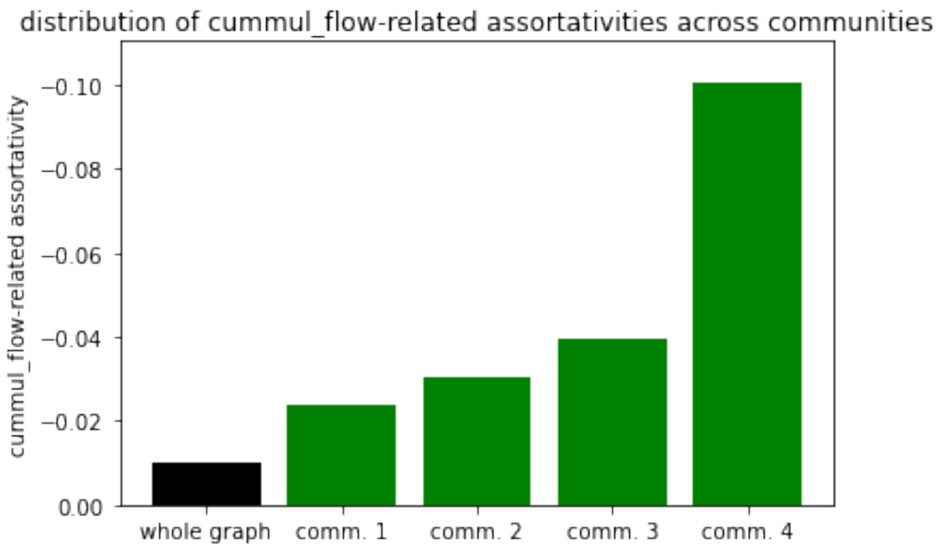
Betweenness-related assortativities in all the different communities

```
In [ ]: plot_assortativities(Und_G, 'betweenness')
```



Carbon flow-related assortativities in all the different communities

```
In [ ]: plot_assortativities(Und_G, 'cummul_flow')
```



A first observation is thats in all cases (except degree assortativity for community 3), the communities show a higher disassortativity than the whole graph, which indicates that species connected within communities tend to show disparities in their properties.

One could think of explaining this by the fact the edges present within the communities only capture relationships between species of different trophic levels that present different properties, while this effect could be attenuated in the whole graph by the interconnectivity between trophic chains that is not captured within communities.

A high disassortativity was observed on degreees within communities. This implies that the topologies captured within communities are the bindings between strongly connected species and weakly connected ones.

There is no betweenness-based assortativity or carbon flow- based assortativity, which indicates that communities are not built around them.

According to those results, it doesn't seem that communities capture clusters of nodes of specific properties but rather topologies in the graph, which results in the degree disassortativity. Community 4 captures a strongly connected node (the detritic compartment DOC) and its weakly connected neighbors (here primary producers), which results in the value of disassortativity. Such topology is also found in communities 1 and 2 where the two other detritic compartments (water DOC and benthic DOC) are found. The presence of those strongly-connected nodes is likely responsible for the values of disassortativities within those communities.

Most species in the third compartment are primary producers and no detritic compartment is present, which likely results in the absence of disassortativity.

Interestingly, the fact that communities are overall more disassortative than the graph itself suggest that edges linking similar nodes are present between communities and not inside of them.

The results of assortativity are consistent with the content of the communtities. The fact that each detritic compartment seems associated to a geographical area (water could be unsalty waters, benthic DOC could be the sea compartment) may indicate that communities were separated based on the location where their species live. But such correlation is hard to establish solely on the labels, and the presence of aquatic species inside of all the communities doesn't seem consistent with this hypothesis.

Another remark is that some cycles exist within this network with carbon recylcling in some circular trophic chains. The existence of both cycles and linear relationships between species of this network could make difficult the analysis of the partition obtained with community detection. Further analyses on the number of cycles would be interesting to understand the relationships captured by communities.

In this project we tried to characterize keystone species in a marine ecosystem and to analyze the structure of this ecosystem.

We showed that our algorithm enables the detection of detritical compartments that distinguish from all other species via their high node-degree and betweeness centrality. We observed that important species in this ecosystem are species that are low in trophic chains. Intuitively, such species are the the main source of carbon for the rest of the ecosystem, which explains their importance.

We hypothesized that the ecosystem is structured around the different levels of trophic chains. We determined the number and characteristics of communities using Louvain algorithm. The low number of communities (4) is in aggreement with our hypothesis, and we tried to attribute each community to a level in the trophic chain according to their characteritics. Nevertheless we observed that the repartition of species does not match our hypothesis. We then tried to characterize the parameters that drive the partition of species using assortativity analyses, which seems to indicate a segregation of communities based on topological structures such as the organisation of species around their detritic compartment.