Name: Yifan Bai

Email: yifan.bai@mail.mcgill.ca

Group: 2

McGill ID: 260562421

Reading from data files: email, powergrid, protein

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

meta = np.loadtxt('../networks/metabolic.edgelist.txt').astype(np.int
64)
power = np.loadtxt('../networks/powergrid.edgelist.txt').astype(np.in
t64)
protein = np.loadtxt('../networks/protein.edgelist.txt').astype(np.in
t64)
```

Get adjacency matrices. No self-loop, multi-edge; undirected graphs used

DISCLAIMER: I followed the instructions word-by-word, and for some questions I went with my own interpretations while keeping original route for comparison purposes

In [2]:
```
def build_matrix(x):
    a = np.zeros((np.amax(x)+1, np.amax(x)+1))
    for i in range(len(x)):
        a[x[i][0]][x[i][1]] = 1
    # undirected
    for i in range(len(a)):
        for j in range(len(a)):
            if i == j:
                a[i][i] = 0
            elif a[i][j] != a[j][i]:
                a[i][j] = 1
                a[j][i] = 1
    return a
```

In [3]:
```
a_meta = build_matrix(meta)
a_power = build_matrix(power)
a_protein = build_matrix(protein)
```

In [4]:
```
def build_matrix_no_simp(x):
    a = np.zeros((np.amax(x)+1, np.amax(x)+1))
    for i in range(len(x)):
        a[x[i][0]][x[i][1]] = 1
    return a
```

In [5]:
```
a_meta_original = build_matrix_no_simp(meta)
a_power_original = build_matrix_no_simp(power)
a_protein_original = build_matrix_no_simp(protein)
```

Question 1

a) degree distribution

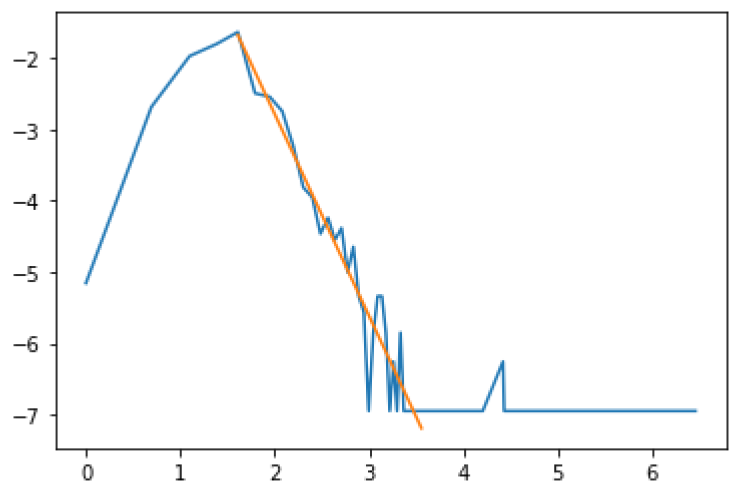
```
In [6]: import matplotlib.pyplot as plt
```

```
In [7]: def get_degree(a, m, begin, end):
            my_list = np.sort(np.sum(a, axis = 1)).tolist()
            freq = {}
            k = []
            v = []
            for items in my_list:
                freq[items] = my_list.count(items)
            for key, value in freq.items():
                k.append(key)
                v.append(value)
            v[:] = [x / sum(v) for x in v]
            x = np.log(k)
            y = np.log(v)
            fit = np.polyfit(x[begin:end], y[begin:end], m)
            print("Coefficients 'a' and 'b' for the linear equation y = ax +
        b: ", fit)
            print("Degree distribution")
            x2 = x[begin:end]
            y2 = fit[0] * x2 + fit[1]
            plt.plot(x, y)
            plt.plot(x2, y2)
            plt.show()
```

Simplified graph

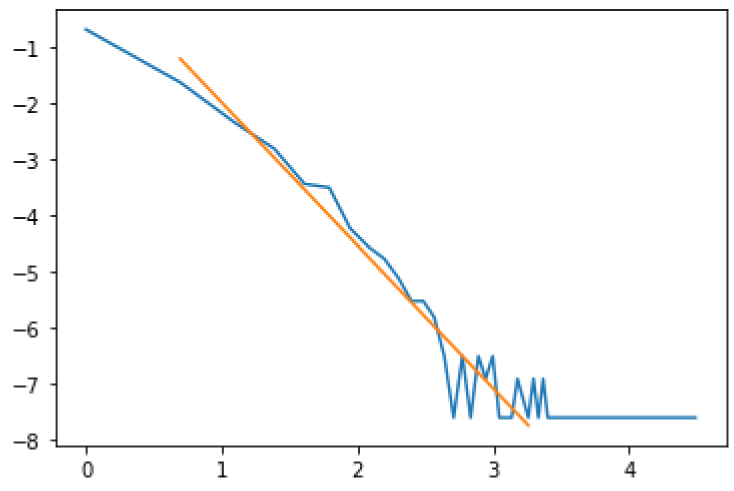
```
In [8]: get_degree(a_meta, 1, 4, 31)
```

```
Coefficients 'a' and 'b' for the linear equation y = ax + b:  [-2.831
83691  2.88065408]
Degree distribution
```



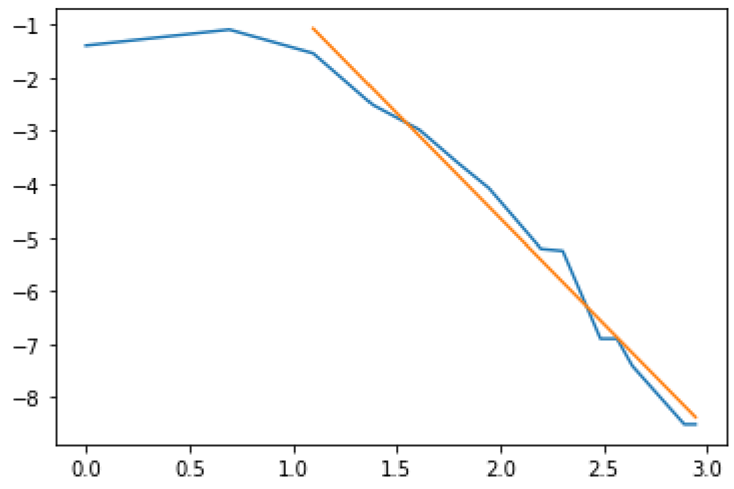
```
In [9]: get_degree(a_protein,1, 2, 25)
```

```
Coefficients 'a' and 'b' for the linear equation y = ax + b:  [-2.545
45078  0.54654605]
Degree distribution

/home/andybai/anaconda3/lib/python3.7/site-packages/ipykernel_launche
r.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
```



```
In [10]: get_degree(a_power, 1,2,16)
```

```
Coefficients 'a' and 'b' for the linear equation y = ax + b:  [-3.953
5314   3.27040433]
Degree distribution
```

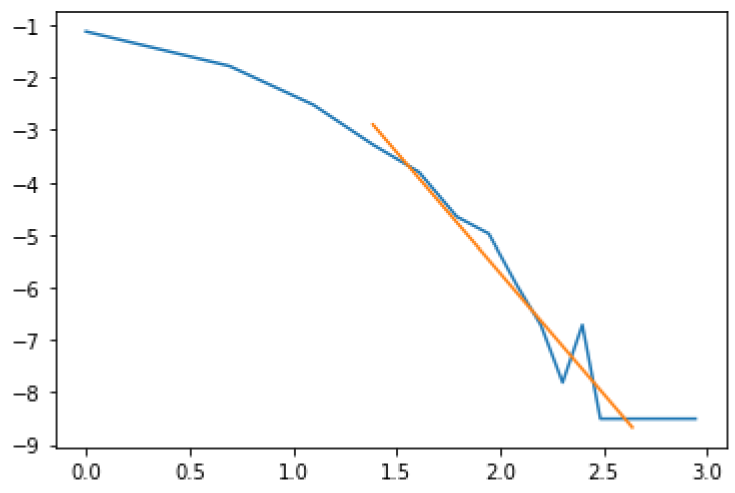


Original graph

```
In [11]: get_degree(a_power_original, 1, 4, 15)
```

```
Coefficients 'a' and 'b' for the linear equation y = ax + b:  [-4.608
09169  3.49206023]
Degree distribution
```

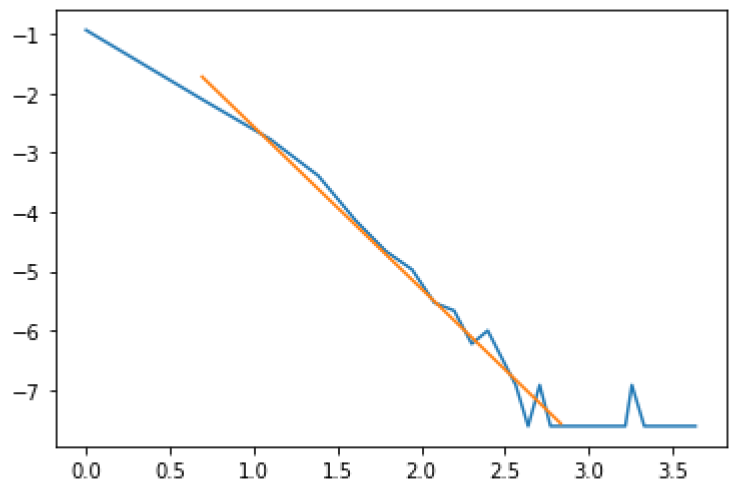
```
/home/andybai/anaconda3/lib/python3.7/site-packages/ipykernel_launche
r.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
```



```
In [12]: get_degree(a_protein_original, 1, 2, 17)
```

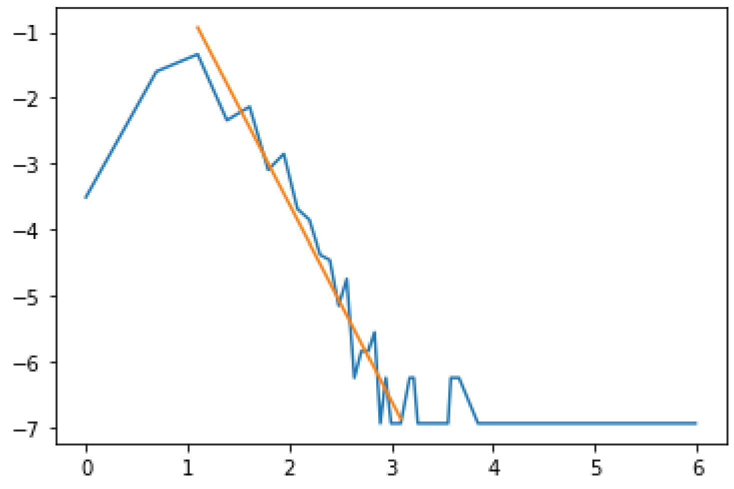
```
Coefficients 'a' and 'b' for the linear equation y = ax + b:  [-2.728
61611  0.1674617 ]
Degree distribution
```

```
/home/andybai/anaconda3/lib/python3.7/site-packages/ipykernel_launche
r.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
```



```
In [13]: get_degree(a_meta_original, 1, 3, 23)
```

```
/home/andybai/anaconda3/lib/python3.7/site-packages/ipykernel_launche
r.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':

Coefficients 'a' and 'b' for the linear equation y = ax + b:  [-2.984
36132  2.34978742]
Degree distribution
```



b) custering coefficient distribution

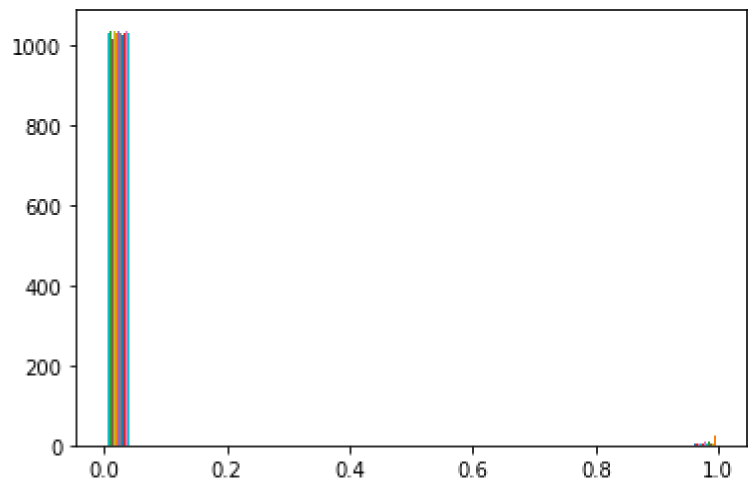
```
In [14]: def clustering_coeff(a):
             c_local = []
             a2 = a * a
             a3 = a2 * a
             a3_diag = np.diag(a3)
             trace = np.trace(a3)
             denom = np.sum(a2) - np.trace(a2)
             print("Total number triangles: ", trace/6)
             print("Global: ", trace/denom)
             degree = np.multiply(np.sum(a, axis = 0), np.sum(a, axis = 0) - 1
         )
             degree2 = np.multiply(np.sum(a, axis = 1), np.sum(a, axis = 1) -
         1)
             for i in range(len(degree)):
                 if degree[i] == 0:
                     c_local.append(0)
                 else:
                     c_local.append(a3_diag[i]/degree[i])
             print("Clustering coefficient, first 10: ", c_local[0:10])
             print("Max clustering coefficient: ", max(c_local))
             print("Min clustering coefficient: ", min(c_local))
             plt.hist(a, bins='auto')
             plt.show()
```

We can see that since the matrices are so sparse, the diagonal of A^3 is a vector of zeros. This leans the clustering coefficients are zeros.

Simplified graphs

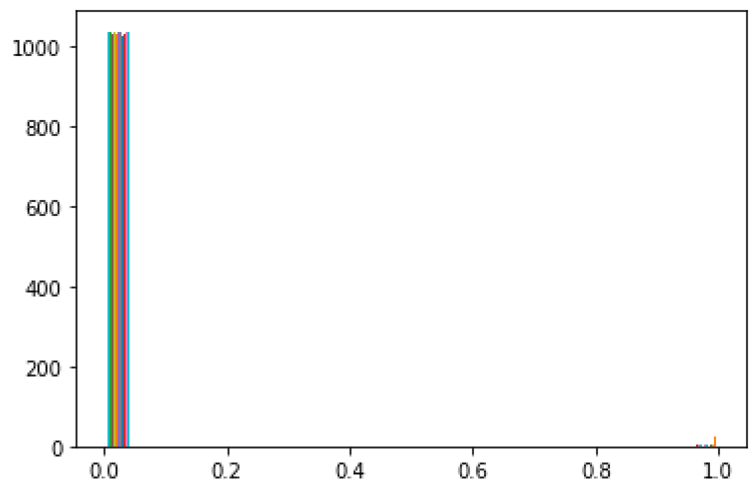
```
In [15]: clustering_coeff(a_meta)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



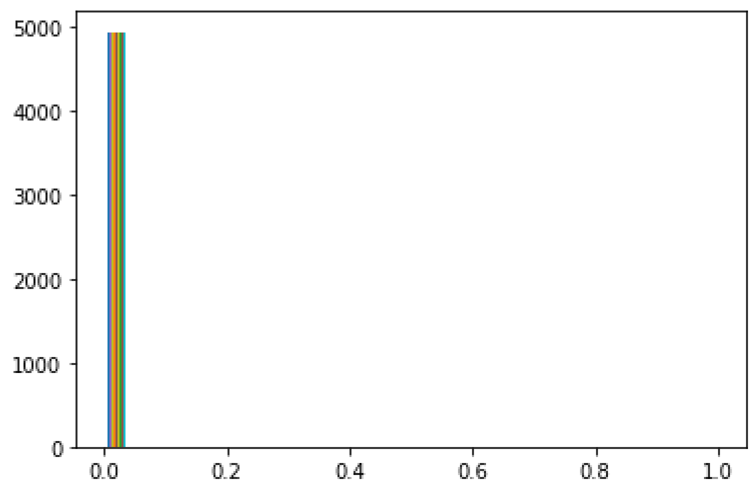
```
In [16]: clustering_coeff(a_meta_original)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



```
In [17]: clustering_coeff(a_power)
```

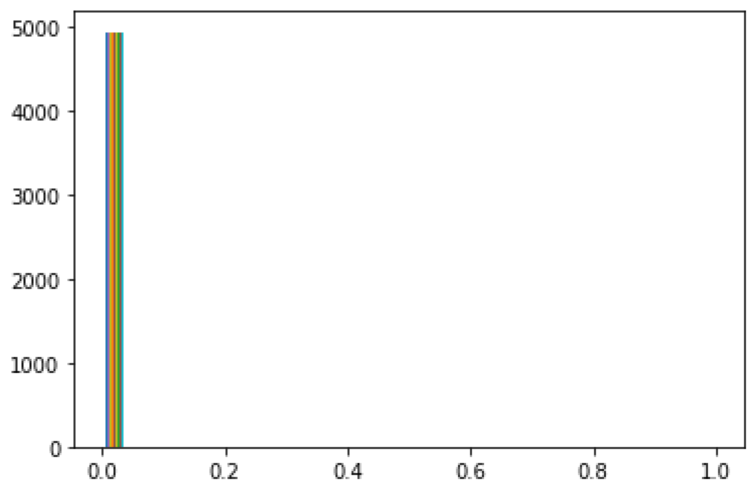
```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0, 0, 0, 0.0, 0, 0, 0.
0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



Original graphs

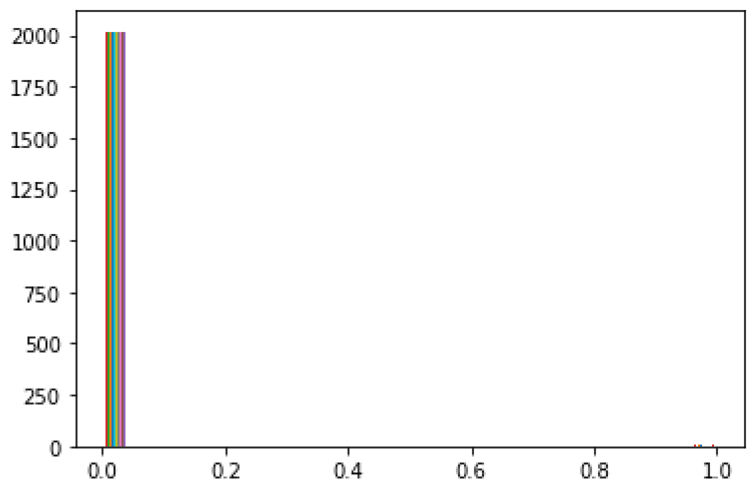
In [18]:
```
clustering_coeff(a_power_original)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0, 0, 0, 0, 0, 0, 0, 0, 0.0, 0]
Max clustering coefficient:  0
Min clustering coefficient:  0
```



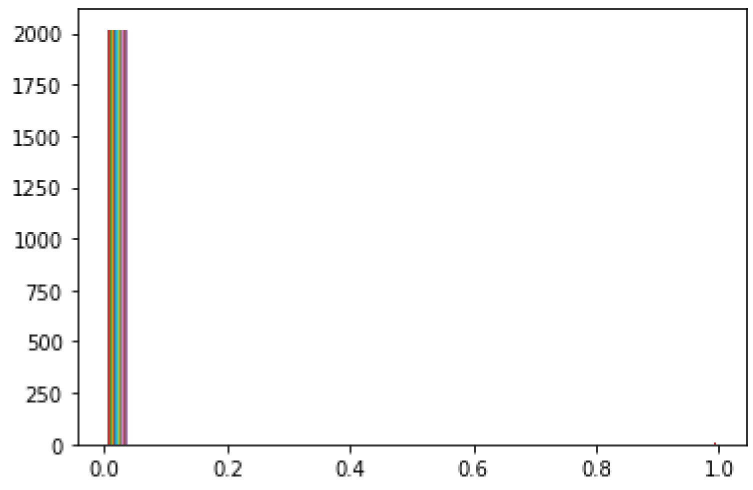
In [19]:
```
clustering_coeff(a_protein)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0, 0, 0, 0, 0, 0, 0, 0, 0.0, 0]
Max clustering coefficient:  0
Min clustering coefficient:  0
```



```
In [20]: clustering_coeff(a_protein_original)
```

```
Total number triangles:  37.5
Global:  0.08317929759704251
Clustering coefficient, first 10:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Max clustering coefficient:  0.5
Min clustering coefficient:  0
```



For c) and d) I only did for the simplified graphs

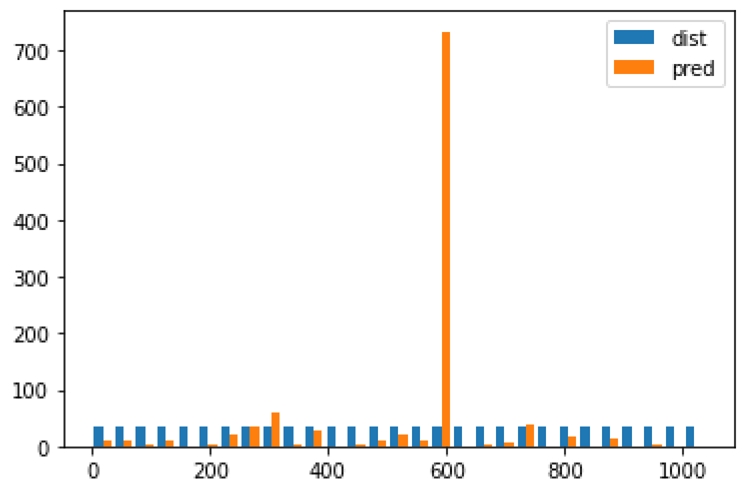
c) shortest paths distribution

```
In [21]: from scipy.sparse import csr_matrix
         from scipy.sparse.csgraph import shortest_path
         from scipy.sparse.csgraph import breadth_first_order
         from scipy.sparse.csgraph import depth_first_order
```

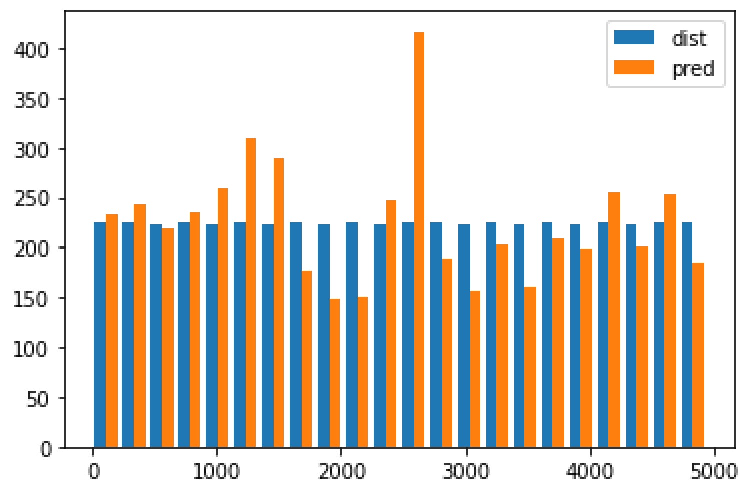
```
In [22]: names = ['metabolism', 'powergrid', 'protein']
         simplified = [a_meta, a_power, a_protein]
         original = [a_meta_original, a_power_original, a_protein_original]
```

```
In [23]: for i in range(len(names)):
             graph = simplified[i]
             graph = csr_matrix(graph)
             dist_matrix = breadth_first_order(csgraph=graph, i_start=0)[0]
             predecessors = breadth_first_order(csgraph=graph, i_start=0)[1]
             predecessors[0] = breadth_first_order(csgraph=graph, i_start=1)[1
         ][0]
             for j in range(len(predecessors)):
                 if predecessors[j] == -9999:
                     predecessors[j] = breadth_first_order(csgraph=graph, i_st
         art=j+1)[1][j]

             print(names[i])
             print("Distribution vector: ", dist_matrix)
             print("Predecessors: ", predecessors)
             print("Histograms for shortest path distributions and predecessor
         s\n")
             plt.hist([dist_matrix, predecessors], bins = 'auto', label=['dis
         t', 'pred'])
             plt.legend(loc='upper right')
             plt.show()
             print("------------------")
         # dist_matrix, predecessors = shortest_path(csgraph=graph, directed=F
         alse, indices=0, return_predecessors=True)
```

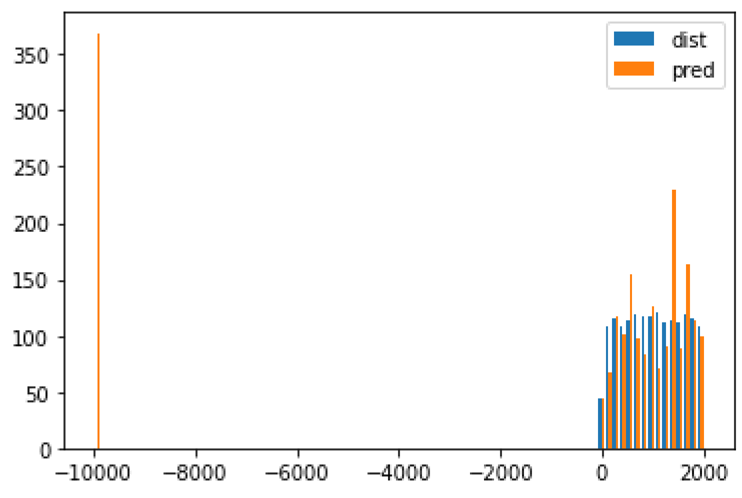
```
metabolism
Distribution vector:  [  0 496 499 ... 238 443 447]
Predecessors:  [589 589 589 ... 589 589 589]
Histograms for shortest path distributions and predecessors
```



```
-----------------
powergrid
Distribution vector:  [   0  386  395 ... 4397 4350 4379]
Predecessors:  [ 395 3586 3583 ... 4929 4933  819]
Histograms for shortest path distributions and predecessors
```



```
-----------------
protein
Distribution vector:  [   0 1050  362 ...  470 1003  116]
Predecessors:  [1050  229  229 ...  806 1982 1637]
Histograms for shortest path distributions and predecessors
```



```
-----------------
```

d) number of connected components

```
In [24]: from scipy.sparse.csgraph import connected_components
```

In [25]:
```
for i in range(len(names)):
    graph = original[i]
    graph = csr_matrix(graph)
    dist_matrix = connected_components(csgraph=graph, directed=False,
return_labels=True)[0]
    predecessors = connected_components(csgraph=graph, directed=False
, return_labels=True)[1]
    print(names[i])
    print('Number of connected components: ', dist_matrix)
    print('Labels: ', predecessors)
    print('Fraction: ',len(np.unique(predecessors))/len(original[i]))
    print("-----------------")
```

```
metabolism
Number of connected components:  1
Labels:  [0 0 0 ... 0 0 0]
Fraction:  0.0009624639076034649
-----------------
powergrid
Number of connected components:  1
Labels:  [0 0 0 ... 0 0 0]
Fraction:  0.00020238818053025704
-----------------
protein
Number of connected components:  185
Labels:  [0 0 0 ... 0 0 0]
Fraction:  0.09167492566897918
-----------------
```

e) eigenvalue distribution

Since we are working with undirected graphs, degree matrix is a diagonal matrix and we can take 'short cuts' to get the Laplacian matrix (L = D - A):

1) Take negative of A[i][j] where i !== j. 2) Direct subtract element wise on the diagonal

In [26]:
```
from scipy.sparse.linalg import eigs
```

In [27]:
```
def eigenvalues(a):
    l = np.zeros(np.shape(a))
    d_i = np.sum(a, axis = 0)
    for i in range(len(a)):
        for j in range(len(a)):
            if i == j:
                l[i][j] = d_i[i] - a[i][j]
            else:
                l[i][j] = -a[i][j]
    vals, vecs = eigs(l, k = 100)
    print("First 100 eigenvalues: \n", vals.real)
    print("Spectral Gap, smallest non-zero eigenvalue for the first 1
00: ", np.amin(vals.real))
```

In [28]:
```
eigenvalues(a_meta)
```

```
First 100 eigenvalues:
 [639.00369299 461.00769966 300.01858498 253.03355538 244.01125795
 140.024371   113.19322153 105.95092568  99.97549584  85.83296667
  83.00698054  84.35134135  67.88033483  61.01881756  54.4144073
  52.3472176   47.95808529  40.31997677  38.7925341   35.82868185
  31.06574579  30.1137395   29.37276965  29.06534315  28.70100681
  28.23300288  27.37510699  27.15566935  26.49210695  25.4748767
  24.97782724  24.67568235  24.51124427  24.42283077  24.05954869
  23.92242291  23.81433488  23.27668833  22.53730101  22.35876669
  22.02287877  21.89057527  21.54350782  21.41948332  21.06071878
  20.85035686  20.48591237  20.37364865  20.05239133  19.71905362
  19.608905    19.00706007  18.93839974  18.73162569  18.48919392
  18.46049692  18.26619604  18.08695384  17.98476779  17.92699474
  17.83812291  17.69688457  17.51475483  17.46135429  17.30464392
  17.22318967  17.15326895  17.18294348  16.8673785   16.81628718
  16.74250819  16.49427838  16.45235344  16.42623394  16.27832818
  16.2299508   16.11084694  15.79375843  15.67896154  15.61101152
  15.56413221  15.48135261  15.3387175   15.22892272  15.05822925
  14.86714526  14.7748687   14.61991023  14.58092543  14.55238427
  14.35936387  14.34952075  14.30414186  14.31058637  14.31449423
  14.07539236  14.03005824  13.97556635  13.86454182  13.82734751]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  13.827
347512074564
```

In [29]:
```
eigenvalues(a_power)
```

```
First 100 eigenvalues:
 [20.10961638 19.12040004 15.53449922 15.09944316 15.06742763 14.3628
3872
 14.44031338 14.43826971 14.12174676 14.09434772 13.99260146 13.36898
903
 13.24934523 13.21895354 13.10166182 13.10060907 12.67055292 12.62922
198
 12.47916386 12.40992733 12.38069949 12.36199878 12.33621813 12.17672
119
 12.19248363 12.19921282 12.07821727 11.76714754 11.69925684 11.60144
844
 11.53268182 11.43900231 11.42075779 11.41498983 11.35582107 11.31966
786
 11.2957789  11.29861273 11.20695083 11.18674019 11.17710596 11.17250
185
 11.16586853 11.14390006 11.13337106 11.09830284 11.06712566 11.04864
904
 10.92360037 10.8631632  10.72548644 10.62953864 10.60360614 10.55202
785
 10.5228627  10.45082532 10.42190636 10.40841961 10.40272777 10.38225
706
 10.35592678 10.33699412 10.33356385 10.31260832 10.27222366 10.23724
402
 10.22183286 10.22740566 10.18366836 10.16249509 10.1238946  10.08133
988
 10.03432053 10.02076735 10.01885597  9.96275101  9.9386608   9.87901
416
  9.84340905  9.82809191  9.80005708  9.64848183  9.64702091  9.61098
323
  9.59156298  9.57337319  9.57335967  9.50198503  9.47834303  9.46844
359
  9.3938688   9.36367353  9.34897219  9.35738015  9.33863969  9.30894
845
  9.29124456  9.24129441  9.22578292  9.21702997]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  9.2170
29967092815
```

In [30]:
```
eigenvalues(a_protein)
```

```
First 100 eigenvalues:
 [90.17842632 82.0563165  80.93315955 51.06632128 45.00888074 43.2449
1536
 38.19367812 31.14190255 30.10368216 30.11833778 29.63420717 27.99609
746
 27.73614027 27.05324546 24.97313848 24.81667957 24.28050261 22.22313
349
 21.39642026 21.02771712 21.01576766 20.27860418 19.79652388 19.46455
393
 19.04617627 18.64419499 18.14425632 17.27041702 17.11152025 16.61834
26
 15.39546503 15.403596   15.19677793 15.16923845 15.07889447 14.49197
983
 14.08251916 13.96831376 13.69029899 13.52937671 13.4864022  13.44390
849
 13.34491274 13.33742644 13.16347371 13.10420386 12.99193218 12.92063
607
 12.70616086 12.58772502 12.50077552 12.31368265 12.02925772 11.85780
264
 11.57302397 11.4255173  11.35220868 11.31698276 11.29967868 11.26114
023
 11.08161508 11.0742745  11.04384324 10.91230838 10.880104   10.86185
407
 10.76025888 10.65170897 10.57400815 10.41047904 10.34391436 10.13549
587
 10.08501948 10.05817254 10.02190807  9.96648631  9.87541734  9.82967
164
  9.72664359  9.67526016  9.52688111  9.51528889  9.4921126   9.40560
267
  9.38481847  9.3407356   9.28653602  9.24283566  9.19056186  9.08151
602
  9.06431483  8.98815457  8.85298914  8.85850807  8.78647354  8.72078
121
  8.7106943   8.64308939  8.57481549  8.55710394]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  8.5571
03935335212
```

f) degree correlations

In [31]:
```
from scipy.stats.stats import pearsonr
```

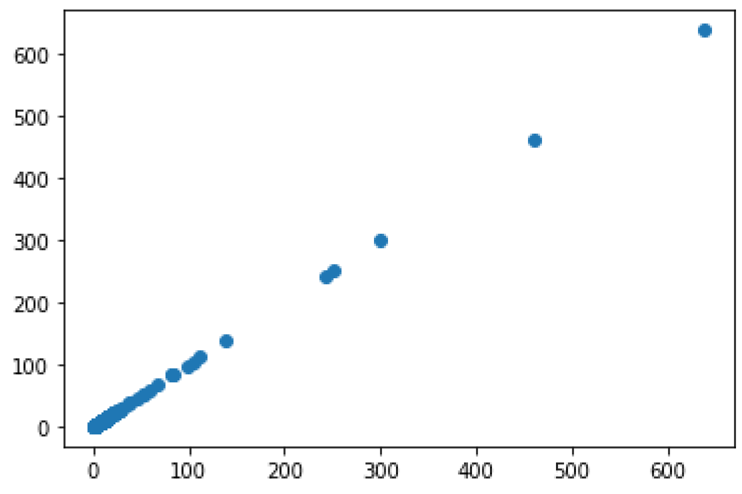
If we keep the simplification stated above, the correlation would be 1. We can investigate how things look like without it.

In [32]:
```
def scatter(a):
    d_i = np.sum(a, axis = 0)
    d_j = np.sum(a, axis = 1)
    plt.scatter(d_i, d_j)
    print("Pearson Correlation Coefficient: ", pearsonr(d_i,d_j)[0])
```

Simplified graphs

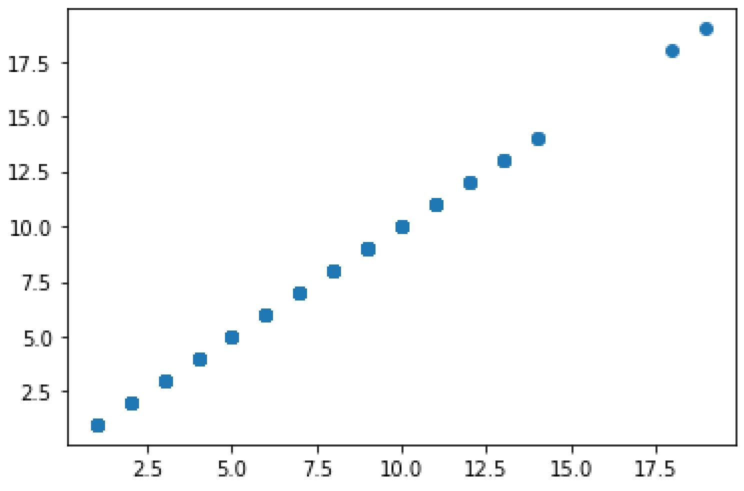
```
In [33]: scatter(a_meta)
```

```
Pearson Correlation Coefficient:  1.0
```



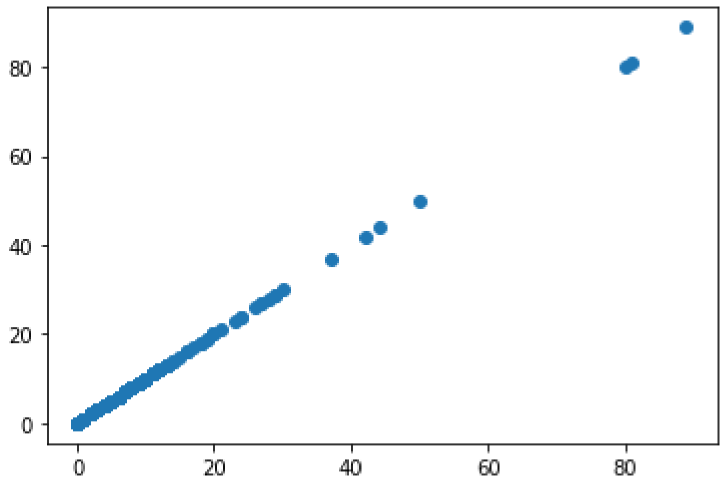
```
In [34]: scatter(a_power)
```

```
Pearson Correlation Coefficient:  1.0
```



```
In [35]: scatter(a_protein)
```

```
Pearson Correlation Coefficient:  0.9999999999999999
```

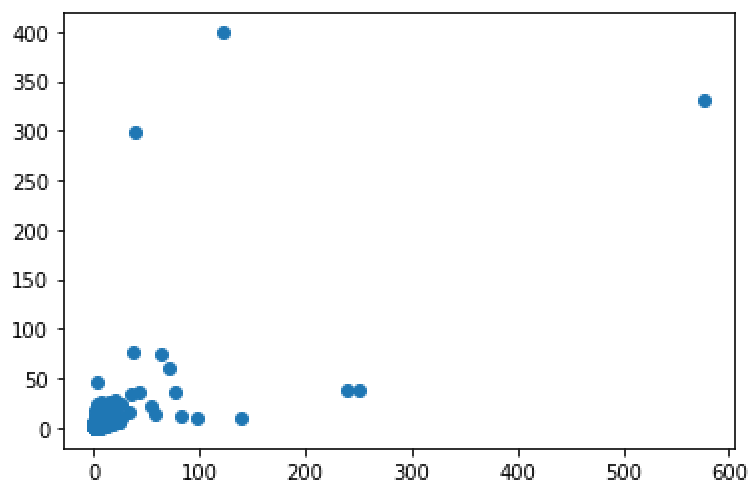


But it would be interesting to see what things are for the original graphs

Original graphs

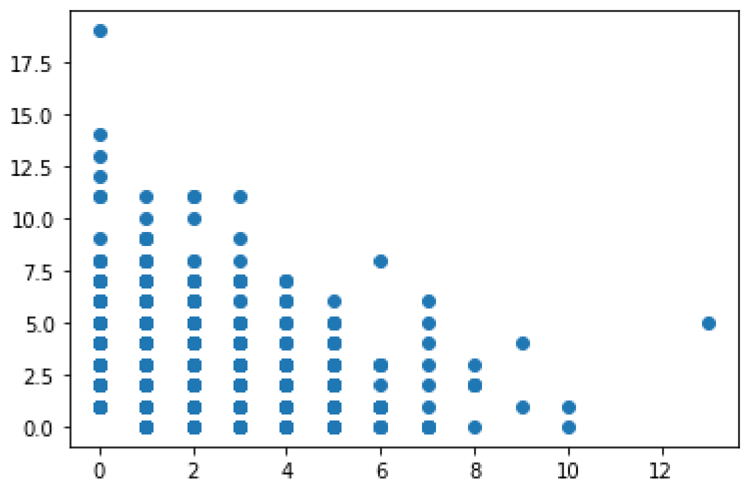
```
In [36]: scatter(a_meta_original)
```

```
Pearson Correlation Coefficient:  0.6332477523312626
```



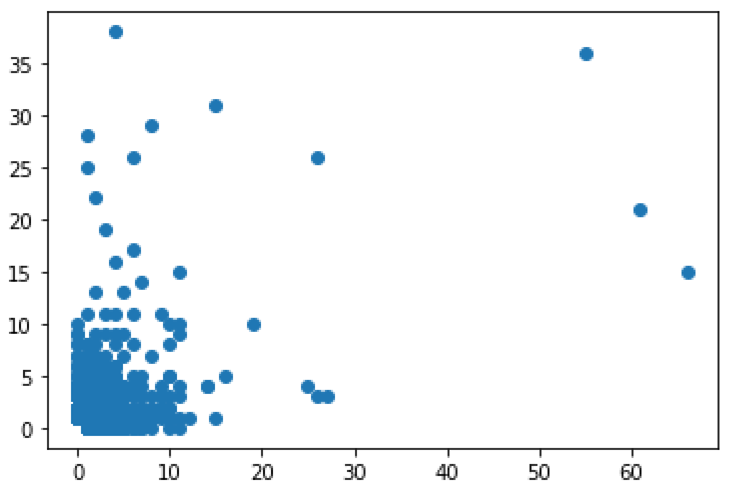
```
In [37]: scatter(a_power_original)
```

```
Pearson Correlation Coefficient:  -0.1770073443062277
```



```
In [38]: scatter(a_protein_original)
```

```
Pearson Correlation Coefficient:  0.4064423366631729
```



As we can see, they are clearly not correlated as before. The reason being that an undirected, self-loop free and single-edged graph is essentially a two-direction simple graph. This would make them to have benign behaviours.

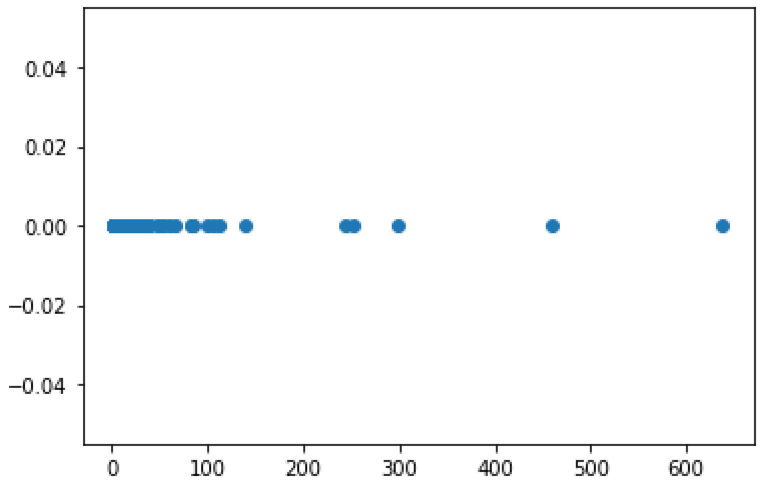
g) degree-clustering coefficient relation

```
In [39]: def d_c(a):
             c_local = []
             a2 = a * a
             a3 = a2 * a
             a3_diag = np.diag(a3)
             degree = np.multiply(np.sum(a, axis = 0), np.sum(a, axis = 0) - 1
         )
             degree2 = np.multiply(np.sum(a, axis = 1), np.sum(a, axis = 1) -
         1)
             for i in range(len(degree)):
                 if degree[i] == 0:
                     c_local.append(0)
                 else:
                     c_local.append(a3_diag[i]/degree[i])
             d_i = np.sum(a, axis = 0)
             plt.scatter(d_i,c_local)
```

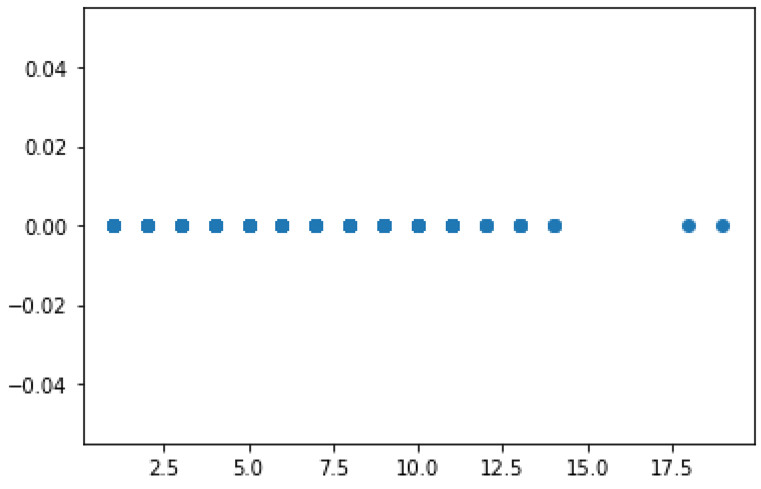
From previous questions, we know that since the clusering coefficients are essentially zero, there would not be any 'definable' correlation.

Simplified graphs

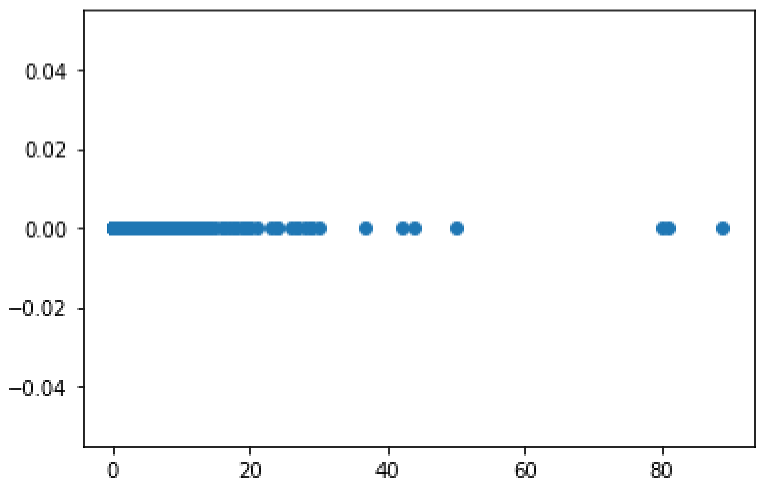
```
In [40]: d_c(a_meta) # matabolism
```



```
In [41]: d_c(a_power)
```

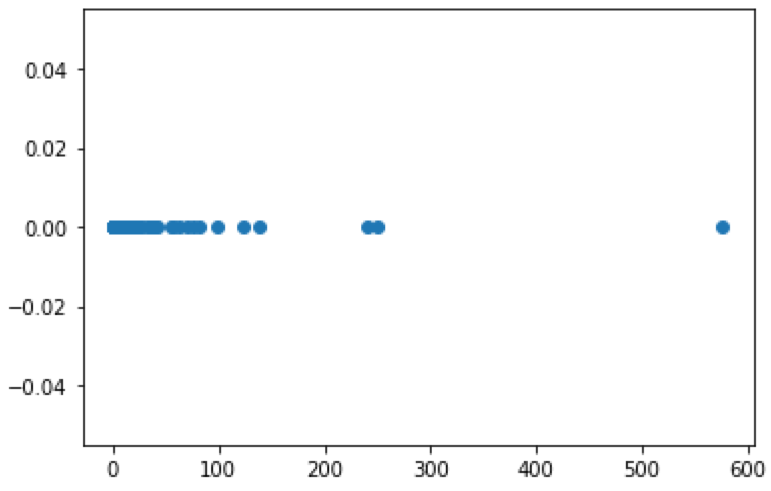


In [42]:
```
d_c(a_protein)
```

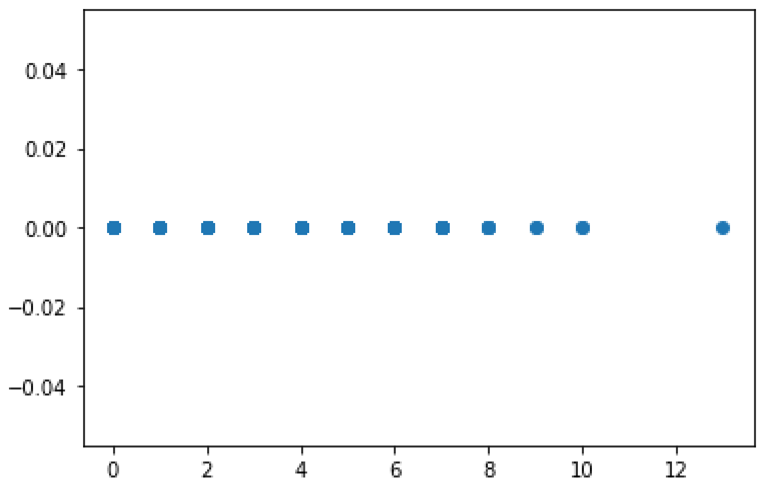


Original graphs

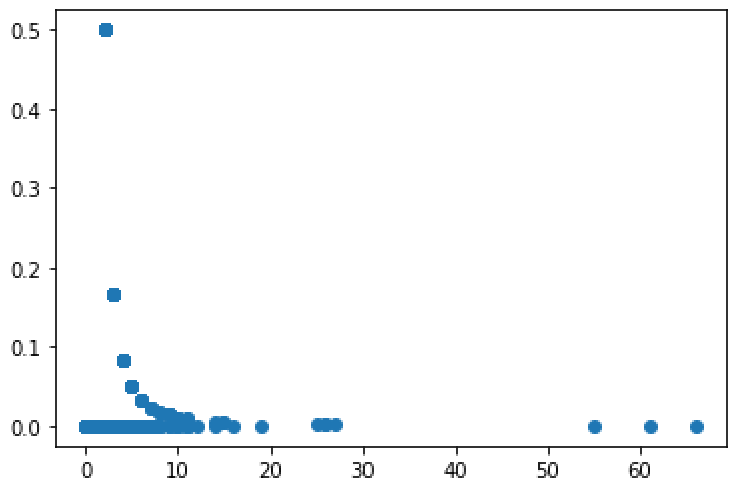
In [43]:
```
d_c(a_meta_original)
```



In [44]:
```
d_c(a_power_original)
```



```
In [45]: d_c(a_protein_original)
```



Question 2

Building Ajacency matrix with n x n shape: n^2 for filling in the entries, then for simplification, n^n access for modification. Therefore O(n^2)

a) Get sum: n operations, initializations: 1 operation, traverse through all sums to build key-value pairs: n operations. Therefore O(n^2)

b) Matrix multiplications: O(n^3), but could use some more advanced as here it is naive. All other operations are similar to the previous questions and are of O(n) or O(n^2). Taking the worst, O(n^3)

c) Since it is BFS/DFS based approach, we visit each node and the complexity is O(n + m) where m is the number of edges in the graph. In an undirected graph, the worst case, m = n(n-1)/2

d) Similar to c) the complexity is O(n + m) where m is the number of edges in the graph. In an undirected graph, the worst case, m = n(n-1)/2

e) To get Laplacian, we need O(n^2) for n x n matrices. Then for eigenvalue the best implementation is O(n^2.376). Taking the worst, O(n^2.376)

f) O(n) summations

g) O(n) for summation to get degree, O(n^3) in the clustering coefficient calculations. Taking the worst, O(n^3)

Part 2

Reading from data files: email, powergrid, protein

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

meta = np.loadtxt('../networks/metabolic.edgelist.txt').astype(np.int
64)
power = np.loadtxt('../networks/powergrid.edgelist.txt').astype(np.in
t64)
protein = np.loadtxt('../networks/protein.edgelist.txt').astype(np.in
t64)
```

If we take the raw text data for graphs

In [2]:
```
# Get sizes of the three networks
# meta
print("Graph 1, edge/node: ", len(meta), np.amax(meta))
# power
print("Graph 2, edge/node: ", len(power), np.amax(power))
# protein
print("Graph 3, edge/node: ", len(protein), np.amax(protein))
```

```
Graph 1, edge/node:  5802 1038
Graph 2, edge/node:  6594 4940
Graph 3, edge/node:  2930 2017
```

With all simplifications, the number of edges, taken as undirected (2 directions), are:

In [3]:
```
def build_matrix(x):
    a = np.zeros((np.amax(x)+1, np.amax(x)+1))
    for i in range(len(x)):
        a[x[i][0]][x[i][1]] = 1
    # undirected
    for i in range(len(a)):
        for j in range(len(a)):
            if i == j:
                a[i][i] = 0
            elif a[i][j] != a[j][i]:
                a[i][j] = 1
                a[j][i] = 1
    return a

a_meta = build_matrix(meta)
a_power = build_matrix(power)
a_protein = build_matrix(protein)
print("Edges for Graphs 1-3: ", np.sum(a_meta)/2, np.sum(a_power)/2,
np.sum(a_protein)/2)
print("Edges for Graphs 1-3, undirected: ", np.sum(a_meta), np.sum(a_
power), np.sum(a_protein))
```

```
Edges for Graphs 1-3:  4741.0 6594.0 2705.0
Edges for Graphs 1-3, undirected:  9482.0 13188.0 5410.0
```

We will pass these valus to the model

File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

Question 3

In [4]:
```
# AB model implementation
import random

def random_subset_with_weights(weights, m):
    mapped_weights = [
        (random.expovariate(w), i)
        for i, w in enumerate(weights)
    ]
    return { i for _, i in sorted(mapped_weights)[:m] }

def barabasi_albert(n, m):
    # initialize with a complete graph on m vertices
    neighbours = [ set(range(m)) - {i} for i in range(m) ]
    degrees = [ m-1 for i in range(m) ]

    for i in range(m, n):
        # stopping criterion if the number of edges is met
        n_neighbours = random_subset_with_weights(degrees, m)

        # add node with back-edges
        neighbours.append(n_neighbours)
        degrees.append(m)

        # add forward-edges
        for j in n_neighbours:
            neighbours[j].add(i)
            degrees[j] += 1
    return neighbours

def barabasi_albert_capped(n, m, deg_cap):
    # initialize with a complete graph on m vertices
    neighbours = [ set(range(m)) - {i} for i in range(m) ]
    degrees = [ m-1 for i in range(m) ]

    for i in range(m, n):
        # stopping criterion if the number of edges is met
        if sum(degrees) >= deg_cap:
            break;
        else:
            n_neighbours = random_subset_with_weights(degrees, m)

            # add node with back-edges
            neighbours.append(n_neighbours)
            degrees.append(m)

            # add forward-edges
            for j in n_neighbours:
                neighbours[j].add(i)
                degrees[j] += 1

    return neighbours
```

The input is number of nodes - n which dictates the node size of the graph, as well as, m - initial number of nodes to start. For the sake of consistency in this section, we start with 3 nodes and try building the graph couting down nodes while respecting the capacity of edges. However, there are some problems to it and the following is used to only match the number of nodes.

In [5]:
```
one = barabasi_albert(1038, 2) # metabolism
two = barabasi_albert(4940, 2) # powergrid
three = barabasi_albert(2017, 2) # protein
```

File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

In [7]:
```
a_one = np.zeros((1038, 1038))
for i in range(len(a_one)):
    for item in list(one[i]):
        a_one[i][item] = 1
```

In [8]:
```
a_two = np.zeros((4940, 4940))
for i in range(len(a_two)):
    for item in list(two[i]):
        a_two[i][item] = 1
```

In [9]:
```
a_three = np.zeros((2017, 2017))
for i in range(len(a_three)):
    for item in list(three[i]):
        a_three[i][item] = 1
```

The following analyses are done with the originally generated graphs

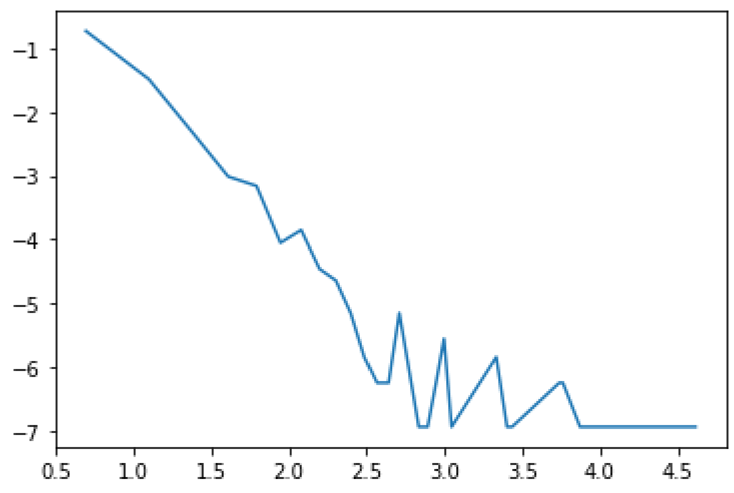
degree distribution

In [10]:
```
one_deg = []
two_deg = []
three_deg = []
for i in range(len(one)):
    one_deg.append(len(one[i]))
for i in range(len(two)):
    two_deg.append(len(two[i]))
for i in range(len(one)):
    three_deg.append(len(three[i]))
```

In [82]:
```
import matplotlib.pyplot as plt

def get_degree(a):
    my_list = a
    freq = {}
    k = []
    v = []
    for items in my_list:
        freq[items] = my_list.count(items)
    for key, value in freq.items():
        k.append(key)
        v.append(value)
    v[:] = [x / sum(v) for x in v]
    x = np.log(k)
    y = np.log(v)
    fit = np.polyfit(x,y, 1)
    print(fit)
    print("Degree distribution")
    plt.plot(x, y)
    plt.show()
```

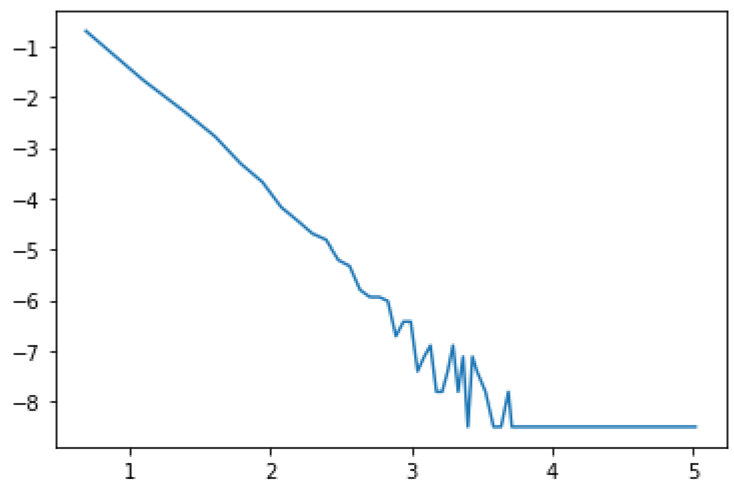
In [52]:
```
get_degree(one_deg)
```

```
[-1.69337888 -0.67269678]
Degree distribution
```



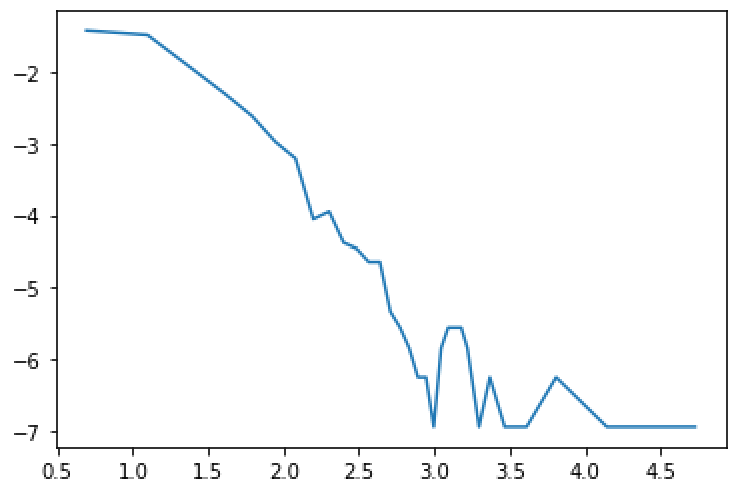
In [53]:
```
get_degree(two_deg)
```

```
[-1.95573416 -0.43836532]
Degree distribution
```



In [54]:
```
get_degree(three_deg)
```

```
[-1.65273012 -0.37599438]
Degree distribution
```



We can fit linear functions very nicely. Here the first number in the list is the slope

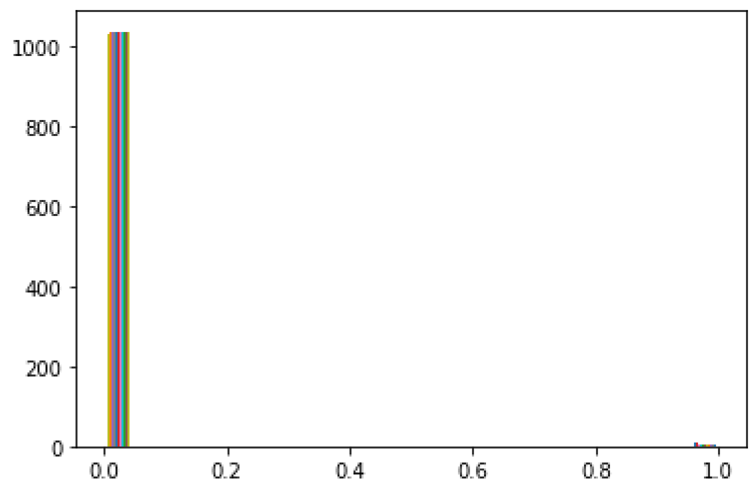
custering coefficient distribution

File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

```
In [15]: def clustering_coeff(a):
             c_local = []
             a2 = a * a
             a3 = a2 * a
             a3_diag = np.diag(a3)
             trace = np.trace(a3)
             denom = np.sum(a2) - np.trace(a2)
             print("Total number triangles: ", trace/6)
             print("Global: ", trace/denom)
             degree = np.multiply(np.sum(a, axis = 0), np.sum(a, axis = 0) - 1
         )
             degree2 = np.multiply(np.sum(a, axis = 1), np.sum(a, axis = 1) -
         1)
             for i in range(len(degree)):
                 if degree[i] == 0:
                     c_local.append(0)
                 else:
                     c_local.append(a3_diag[i]/degree[i])
             print("Clustering coefficient, first 10: ", c_local[0:10])
             print("Max clustering coefficient: ", max(c_local))
             print("Min clustering coefficient: ", min(c_local))
             plt.hist(a, bins='auto')
             plt.show()
```

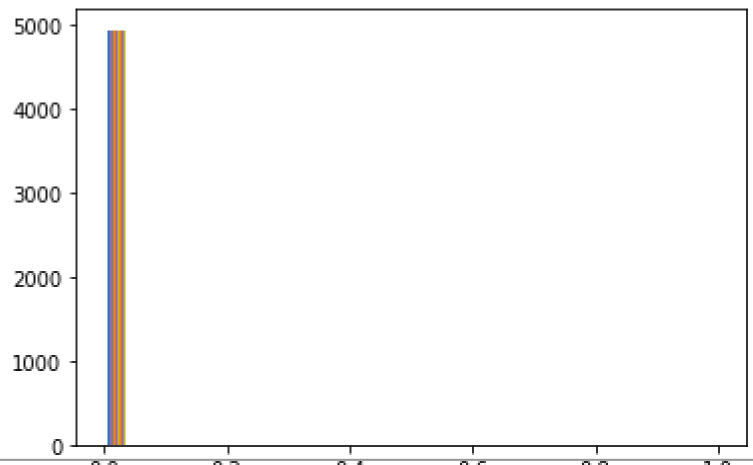
```
In [16]: clustering_coeff(a_one)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



```
In [17]: clustering_coeff(a_two)
```

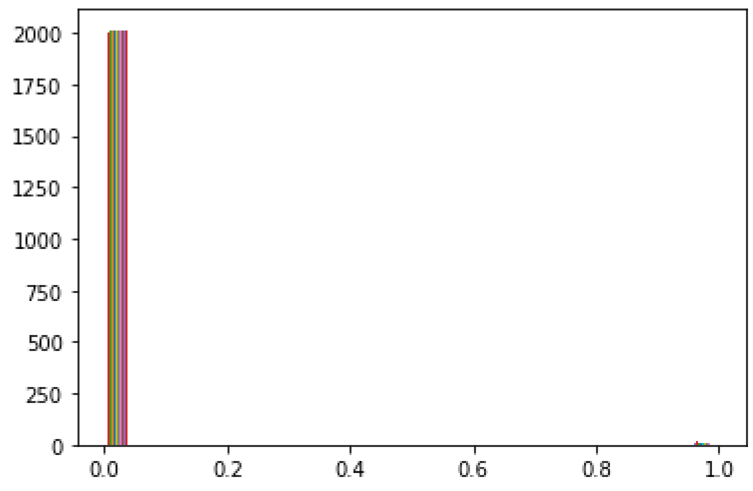
```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

```
In [18]: clustering_coeff(a_three)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



Since traces are all zero, the values are all zero.

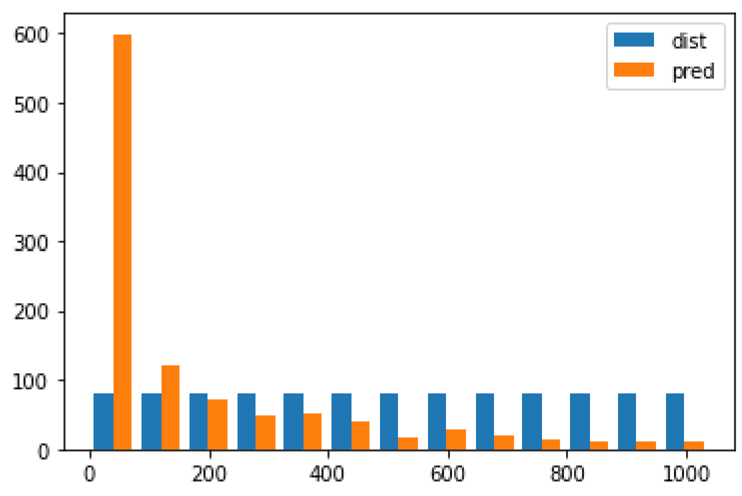
shortest paths distribution number of connected components

```
In [19]: from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import shortest_path
from scipy.sparse.csgraph import breadth_first_order
from scipy.sparse.csgraph import depth_first_order
from scipy.sparse.csgraph import connected_components

names = ['One', 'Two', 'Three']
mats = [a_one, a_two, a_three]
```

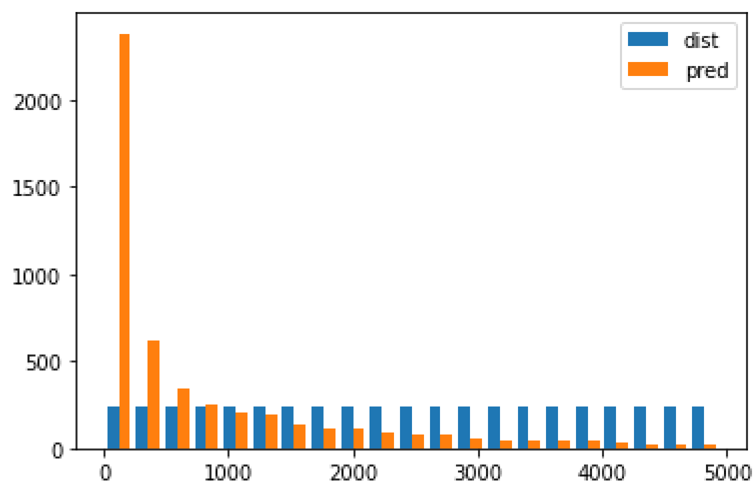
File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

```
In [20]: for i in range(len(names)):
             graph = mats[i]
             graph = csr_matrix(graph)
             dist_matrix = breadth_first_order(csgraph=graph, i_start=0)[0]
             predecessors = breadth_first_order(csgraph=graph, i_start=0)[1]
             predecessors[0] = breadth_first_order(csgraph=graph, i_start=1)[1
         ][0]
             for j in range(len(predecessors)):
                 if predecessors[j] == -9999:
                     predecessors[j] = breadth_first_order(csgraph=graph, i_st
         art=j+1)[1][j]

             print(names[i])
             print("Distribution vector: ", dist_matrix)
             print("Predecessors: ", predecessors)
             print("Histograms for shortest path distributions and predecessor
         s\n")
             plt.hist([dist_matrix, predecessors], bins = 'auto', label=['dis
         t', 'pred'])
             plt.legend(loc='upper right')
             plt.show()
             print("-----------------")
```

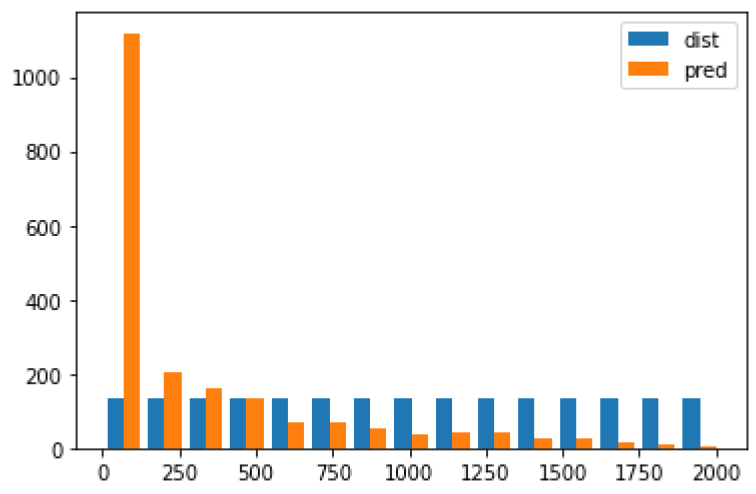
```
One
Distribution vector:  [   0    1    2 ... 1007  954  841]
Predecessors:  [ 1  0  0 ...  2 48 24]
Histograms for shortest path distributions and predecessors
```



```
-----------------
Two
Distribution vector:  [   0    1    2 ... 3156 4067 4236]
Predecessors:  [   1    0    0 ... 1724 4520 1554]
Histograms for shortest path distributions and predecessors
```



```
-----------------
Three
Distribution vector:  [   0    1    2 ... 1561 1872 2005]
Predecessors:  [   1    0    0 ...   63 1059   52]
Histograms for shortest path distributions and predecessors
```



```
-----------------
```

The predecessor vector values are much more skewed towards lower end while those of distribution matrices are very well spread. Some nodes are much more connected while the rest are much less. It confirms the 'rich get richer' conundrum.

```
In [21]: for i in range(len(names)):
             graph = mats[i]
             graph = csr_matrix(graph)
             dist_matrix = connected_components(csgraph=graph, directed=False,
         return_labels=True)[0]
             predecessors = connected_components(csgraph=graph, directed=False
         , return_labels=True)[1]
             print(names[i])
             print('Number of connected components: ', dist_matrix)
             print('Labels: ', predecessors)
             print('Fraction: ',len(np.unique(predecessors))/len(mats[i]))
             print("-----------------")
```

```
One
Number of connected components:  1
Labels:  [0 0 0 ... 0 0 0]
Fraction:  0.0009633911368015414
-----------------
Two
Number of connected components:  1
Labels:  [0 0 0 ... 0 0 0]
Fraction:  0.00020242914979757084
-----------------
Three
Number of connected components:  1
Labels:  [0 0 0 ... 0 0 0]
Fraction:  0.0004957858205255329
-----------------
```

There is only 1 in GCC

eigenvalue distribution

```
In [22]: from scipy.sparse.linalg import eigs

         def eigenvalues(a):
             l = np.zeros(np.shape(a))
             d_i = np.sum(a, axis = 0)
             for i in range(len(a)):
                 for j in range(len(a)):
                     if i == j:
                         l[i][j] = d_i[i] - a[i][j]
                     else:
                         l[i][j] = -a[i][j]
             vals, vecs = eigs(l, k = 100)
             print("First 100 eigenvalues: \n", vals.real)
             print("Spectral Gap, smallest non-zero eigenvalue for the first 1
         00: ", np.amin(vals.real))
```

In [23]:
```
eigenvalues(a_one)
```

```
First 100 eigenvalues:
 [102.05521777  61.36122693  57.86836162  49.07238684  45.11660979
  44.2017448   42.77100739  41.98860484  32.37518507  31.44372381
  29.07574337  28.75553207  28.64773623  22.09371301  21.85017486
  21.15381948  20.97266458  20.73078091  20.11543885  19.93094085
  18.74543295  18.32583458  16.23782461  16.17735616  16.15975529
  15.97740865  15.93152314  15.87845801  14.78861499  14.29142139
  14.17753208  14.01731904  13.62329072  13.34205331  12.89776933
  12.72027018  12.51305684  12.41733904  11.97383259  11.50802843
  11.44386277  11.41085756  11.27700435  11.23476915  11.21955105
  11.20737659  11.02925909  10.95768719  10.8608926   10.76229435
  10.58997014  10.57383329  10.54564365  10.41093942  10.38434219
  10.30828258  10.24257019  10.25832344  10.16796867   9.99540648
   9.94570396   9.84678326   9.65067775   9.51678307   9.50188927
   9.3708234    9.29675108   9.19072595   9.15730719   9.13690726
   9.0968882    9.02106231   8.9625121    8.92082499   8.86020021
   8.83772555   8.74371865   8.6947344    8.5716959    8.53915842
   8.53586607   8.45667118   8.42430757   8.38719557   8.33413545
   8.31026274   8.29147675   8.21414261   8.19074608   8.07949163
   7.97457743   7.9555335    7.94762125   7.88633205   7.87684497
   7.85310645   7.84552926   7.82661684   7.79959204   7.78394185]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  7.7839
41850853903
```

In [24]:
```
eigenvalues(a_two)
```

```
First 100 eigenvalues:
 [152.28741692 147.92502376 137.95699722 122.05861395  89.07973713
  76.22864176  71.84359955  71.00529296  70.0362286   65.03296383
  61.11357143  56.02195568  53.89368391  53.02920831  49.24851365
  43.06236268  42.39751384  41.13256894  41.10722315  38.68984935
  37.21677532  35.08509859  35.02194241  33.08393924  32.98897384
  32.89206772  32.19880564  32.08665768  31.99771367  32.04058469
  31.00196399  30.18459003  30.09593589  30.09083048  30.13667162
  29.22604275  29.05378923  28.16856588  27.95756567  28.09334904
  28.05730367  28.04925715  27.43900817  27.14154552  27.07965638
  27.00411354  25.92878897  25.16919265  24.77828817  24.51500932
  24.36252498  24.27119707  24.14756274  23.83009357  23.15105232
  23.01959744  22.98044815  22.6888324   22.27742897  22.00699523
  21.87664315  21.94352428  21.19670631  21.02120264  20.99733042
  20.96968989  20.91093656  20.85150011  20.87926082  20.42491522
  20.40515024  20.34628917  20.1666042   20.01277053  20.00660173
  19.93953055  19.90445324  19.45923554  19.23160997  19.18905887
  19.0586045   18.98915216  18.75577146  18.43584559  18.41877568
  18.38516363  18.3343593   18.24946986  18.15704951  18.11939265
  18.06937602  17.86050097  17.8213011   17.84478647  17.72502407
  17.71713087  17.73564134  17.49527449  17.3594119   17.30931229]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  17.309
31229469935
```

File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

In [25]:
```
eigenvalues(a_three)
```

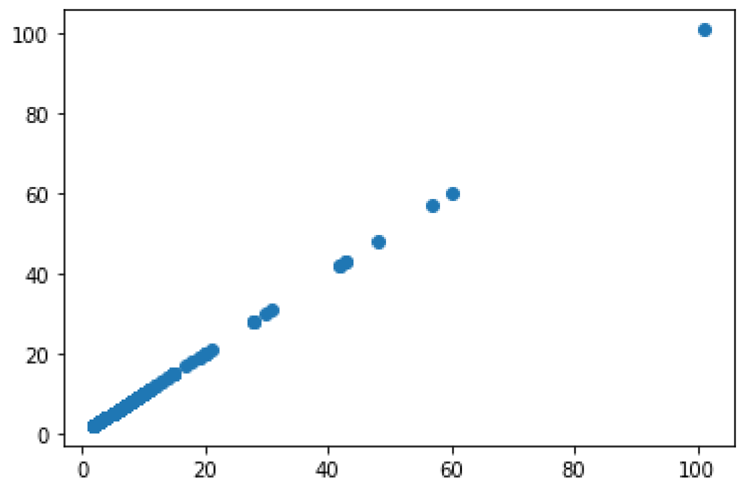
```
First 100 eigenvalues:
 [114.10412894  94.59764901  92.55919976  75.03194834  64.99770634
  63.9496844   46.13136637  45.9760032   38.10158711  33.01392355
  31.17547757  29.29316351  28.05900879  26.08968622  26.12946407
  25.84256819  25.27550791  25.17478965  25.09005249  24.92999406
  23.29704559  23.17492514  23.05350258  22.97229845  22.84187081
  22.21456998  21.14009366  21.07714197  20.20649956  19.99417416
  19.02266207  18.72642564  18.30250017  18.16157405  18.12070426
  17.84884092  17.76038255  17.17022465  16.79418879  16.60380164
  16.30358846  16.26203004  16.11891002  15.65152054  15.59395715
  15.53660465  15.18760825  15.1411815   15.10241871  15.0657644
  14.98621061  14.85731408  14.83266387  14.55003749  14.37939937
  14.35540686  14.34210221  14.27190181  14.23431487  14.18066304
  14.09955352  14.00523444  13.77977282  13.51324822  13.4893056
  13.45613793  13.34609875  13.24722714  13.2171924   13.15518968
  13.00944506  12.96105598  12.83764134  12.79915008  12.64770021
  12.61136301  12.56849994  12.51586337  12.46767747  12.38658176
  12.36479401  12.33493046  12.15296258  12.14191015  12.08001627
  11.99325371  11.80925691  11.73656481  11.7092937   11.64238821
  11.58413329  11.47871565  11.44802352  11.43473579  11.40350687
  11.37074522  11.19817393  11.16706812  11.15228445  11.059209  ]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  11.059
20900217497
```

degree correlations

In [26]:
```
from scipy.stats.stats import pearsonr

def scatter(a):
    d_i = np.sum(a, axis = 0)
    d_j = np.sum(a, axis = 1)
    plt.scatter(d_i, d_j)
    print("Pearson Correlation Coefficient: ", pearsonr(d_i,d_j)[0])
```

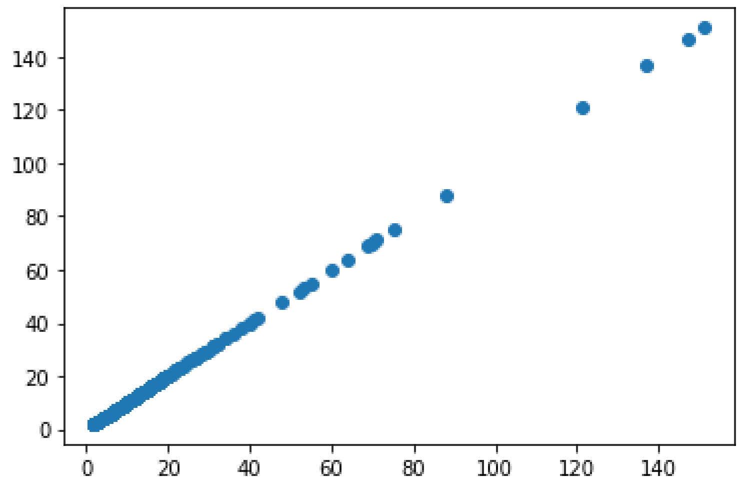
In [27]:
```
scatter(a_one)
```

```
Pearson Correlation Coefficient:  1.0
```



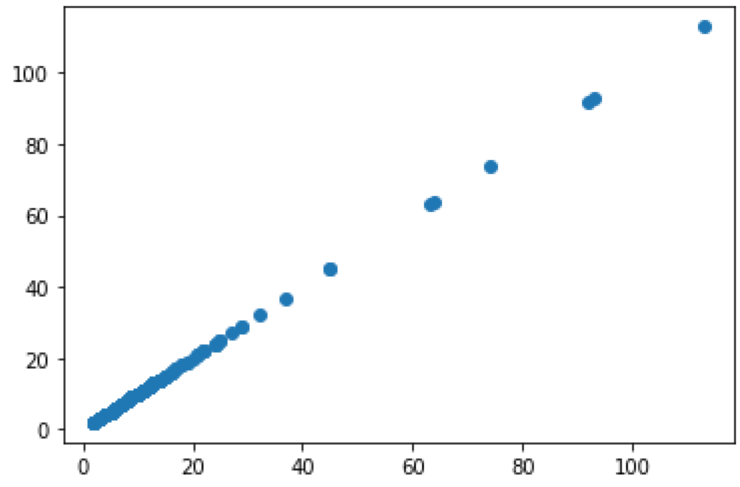
```
In [28]: scatter(a_two)
```

```
Pearson Correlation Coefficient:  0.9999999999999956
```



```
In [29]: scatter(a_three)
```

```
Pearson Correlation Coefficient:  0.9999999999999976
```

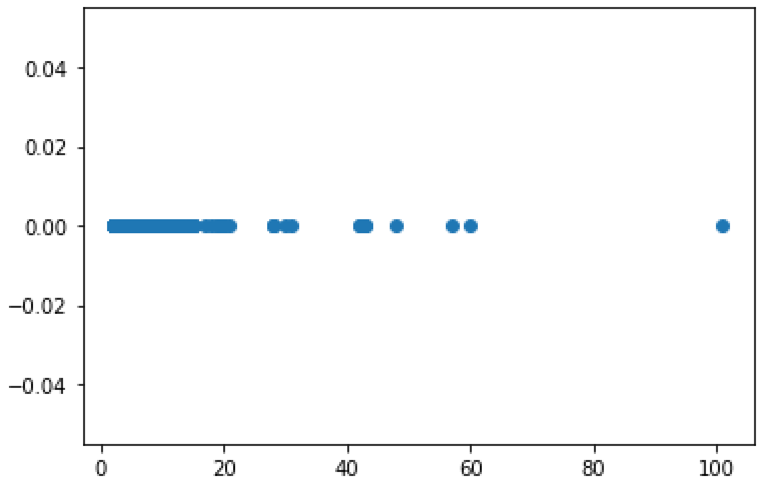


Very high correlation irrespective of number of nodes/edges.

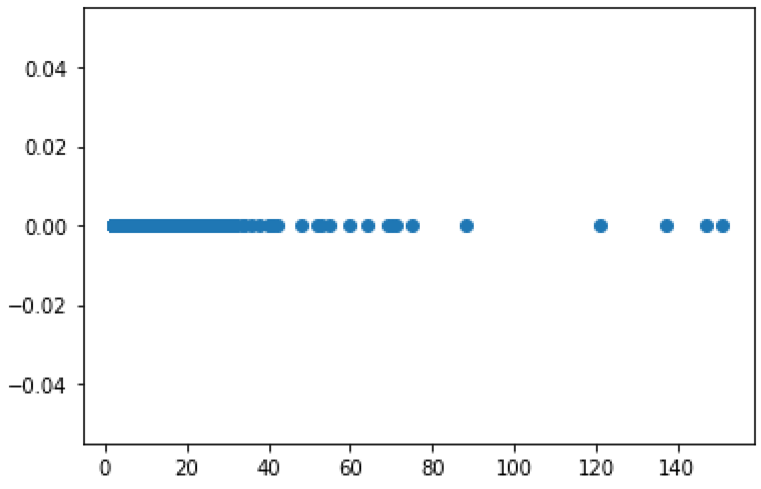
degree-clustering coefficient relation

```
In [30]: def d_c(a):
             c_local = []
             a2 = a * a
             a3 = a2 * a
             a3_diag = np.diag(a3)
             degree = np.multiply(np.sum(a, axis = 0), np.sum(a, axis = 0) - 1
         )
             degree2 = np.multiply(np.sum(a, axis = 1), np.sum(a, axis = 1) -
         1)
             for i in range(len(degree)):
                 if degree[i] == 0:
                     c_local.append(0)
                 else:
                     c_local.append(a3_diag[i]/degree[i])
             d_i = np.sum(a, axis = 0)
             plt.scatter(d_i,c_local)
```

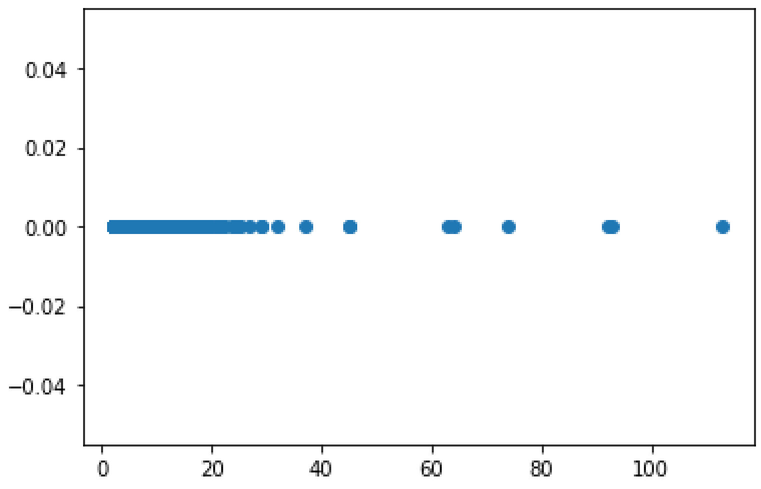
In [31]:
```
d_c(a_one)
```



In [32]:
```
d_c(a_two)
```



In [33]:
```
d_c(a_three)
```



This would be zero since clustering coefficients are zero.

Extension

Try another randomization method for the smallest graph

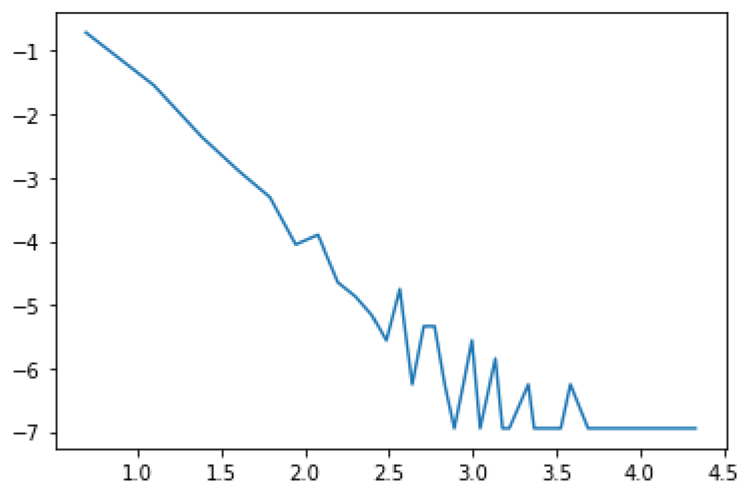
In [74]:
```
def random_subset_with_weights_pareto(weights, m):
    mapped_weights = [
        (random.paretovariate(w), i)
        for i, w in enumerate(weights)
    ]
    return { i for _, i in sorted(mapped_weights)[:m] }

def barabasi_albert_one(n, m):
    # initialize with a complete graph on m vertices
    neighbours = [ set(range(m)) - {i} for i in range(m) ]
    degrees = [ m-1 for i in range(m) ]

    for i in range(m, n):
        # stopping criterion if the number of edges is met
        n_neighbours = random_subset_with_weights_beta(degrees, m)

        # add node with back-edges
        neighbours.append(n_neighbours)
        degrees.append(m)

        # add forward-edges
        for j in n_neighbours:
            neighbours[j].add(i)
            degrees[j] += 1
    return neighbours
```

In [75]:
```
four = barabasi_albert_one(1038, 2)
a_four = np.zeros((1038, 1038))
for i in range(len(a_four)):
    for item in list(four[i]):
        a_four[i][item] = 1
```

In [76]:
```
four_deg = []
for i in range(len(four)):
    four_deg.append(len(four[i]))
```

In [83]:
```
get_degree(four_deg)
```

```
[-1.99588874  0.00895811]
Degree distribution
```

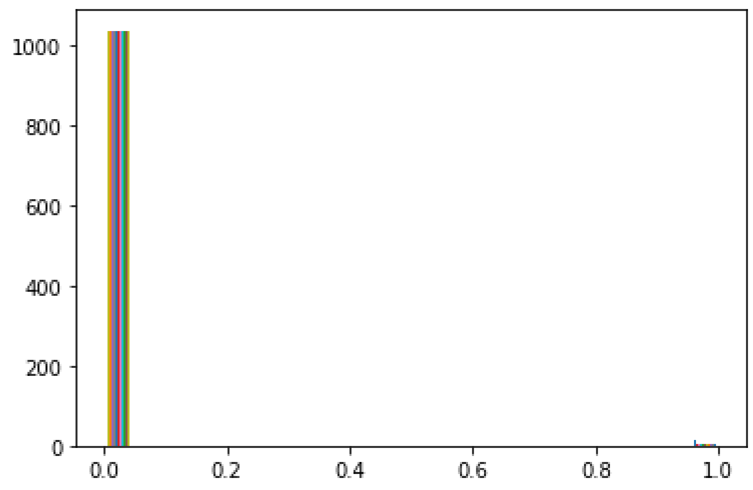


There are only slight differences in coefficients but overall the trends are same. It means the randomization process won't revolutionarily change things.

File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

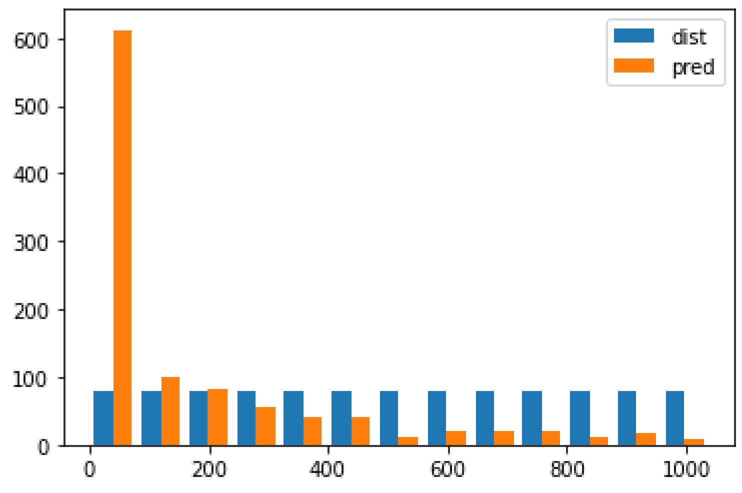
In [84]:
```
clustering_coeff(a_four)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



In [85]:
```
names2 = ['Four']
mats2 = [a_four]

for i in range(len(names2)):
    graph = mats2[i]
    graph = csr_matrix(graph)
    dist_matrix = breadth_first_order(csgraph=graph, i_start=0)[0]
    predecessors = breadth_first_order(csgraph=graph, i_start=0)[1]
    predecessors[0] = breadth_first_order(csgraph=graph, i_start=1)[1
][0]
    for j in range(len(predecessors)):
        if predecessors[j] == -9999:
            predecessors[j] = breadth_first_order(csgraph=graph, i_st
art=j+1)[1][j]

    print(names2[i])
    print("Distribution vector: ", dist_matrix)
    print("Predecessors: ", predecessors)
    print("Histograms for shortest path distributions and predecessor
s\n")
    plt.hist([dist_matrix, predecessors], bins = 'auto', label=['dis
t', 'pred'])
    plt.legend(loc='upper right')
    plt.show()
    print("-----------------")
```

```
Four
Distribution vector:  [  0   1   2 ... 958 798 937]
Predecessors:  [ 1  0  0 ... 31  2 10]
Histograms for shortest path distributions and predecessors
```



```
-----------------
```

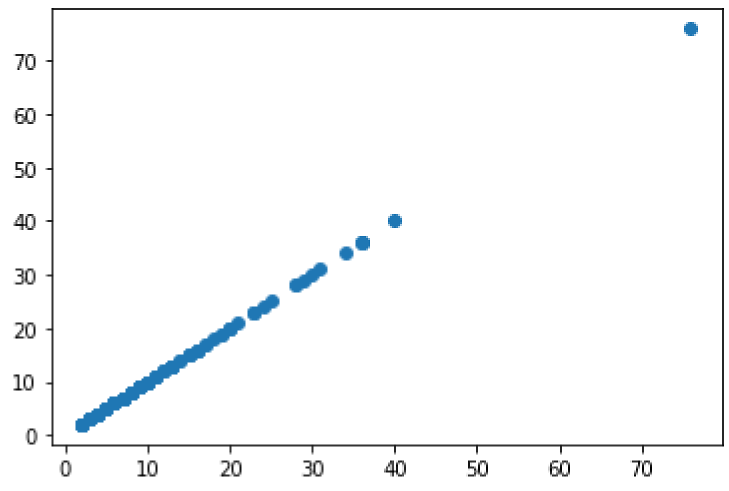
File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

```
In [86]: eigenvalues(a_four)
```

```
First 100 eigenvalues:
 [77.11601743 41.35560575 37.8957354  36.28451538 35.34951753 32.1376
2735
 31.25539453 29.99519244 28.9276927  28.85288126 26.39865573 24.76792
543
 24.51949542 24.19625225 24.257059   22.1593592  21.97197673 21.10656
013
 20.5775282  20.44401837 20.04823442 19.94833797 19.43755713 18.55620
201
 17.54980957 17.37225521 17.31310769 17.21826584 17.13114483 16.33220
079
 16.28622731 16.13130556 15.93774661 15.89802961 15.75096667 15.29173
983
 14.91930542 14.23804544 14.24445417 14.01515158 13.95870142 13.82491
531
 13.68762551 13.66154511 13.60787549 13.41409974 13.21815829 13.09764
242
 12.97737171 12.90915195 12.85873631 12.48049328 12.05452233 11.87117
486
 11.83368055 11.55218661 11.34843658 11.06583203 10.93845173 10.89860
018
 10.85302662 10.81535769 10.55609542 10.48864618 10.46622846 10.40635
596
 10.37920106 10.26342745 10.12947982 10.11268739  9.96281579  9.93472
573
  9.84958445  9.53189286  9.48013672  9.39637294  9.36781137  9.34965
61
  9.23555128  9.1299522   9.10724747  9.09491548  9.06219999  9.00106
713
  8.94986013  8.94189165  8.90653971  8.85336457  8.78107442  8.64518
224
  8.62123524  8.54586391  8.50356547  8.4111222   8.42349275  8.27893
706
  8.26340347  8.22577388  8.15598959  8.09363693]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  8.0936
3692988137
```

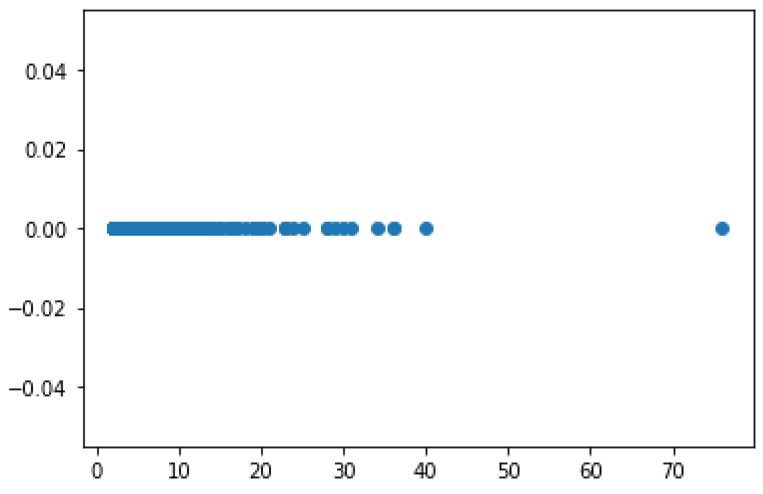
```
In [87]: scatter(a_four)
```

```
Pearson Correlation Coefficient:  0.9999999999999998
```



File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

```
In [88]: d_c(a_four)
```



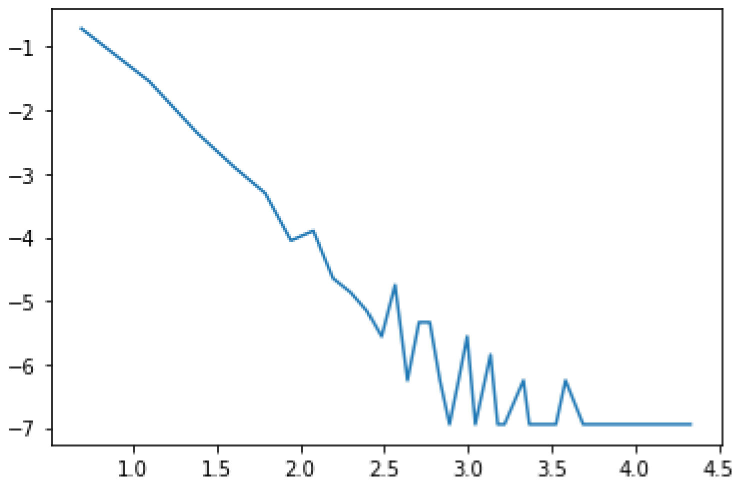
The rest confirm the same story

Try with more initial points. This will increase number of edges.

```
In [93]: five = barabasi_albert(1038, 5)
         a_five = np.zeros((1038, 1038))
         for i in range(len(a_five)):
             for item in list(five[i]):
                 a_five[i][item] = 1
         five_deg = []
         for i in range(len(five)):
             five_deg.append(len(five[i]))
```

```
In [94]: get_degree(four_deg)
```

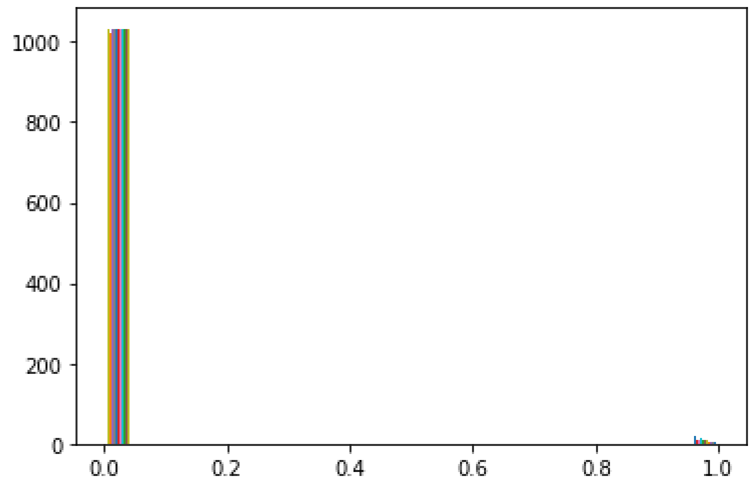
```
[-1.99588874  0.00895811]
Degree distribution
```



File failed to load: file:///home/andybai/Desktop/COMP-596-Network-Science/a1/Q3_files/extensions/MathZoom.js

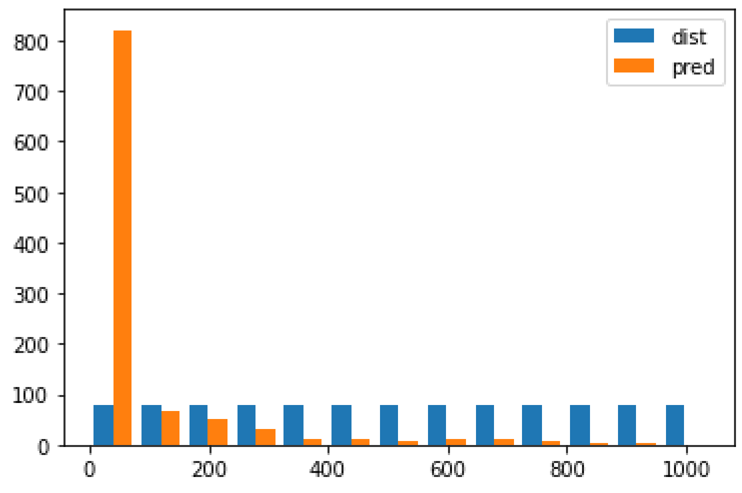
In [95]:
```
clustering_coeff(a_five)
```

```
Total number triangles:  0.0
Global:  0.0
Clustering coefficient, first 10:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0]
Max clustering coefficient:  0.0
Min clustering coefficient:  0.0
```



In [96]:
```
names3 = ['Five']
mats3 = [a_five]

for i in range(len(names3)):
    graph = mats3[i]
    graph = csr_matrix(graph)
    dist_matrix = breadth_first_order(csgraph=graph, i_start=0)[0]
    predecessors = breadth_first_order(csgraph=graph, i_start=0)[1]
    predecessors[0] = breadth_first_order(csgraph=graph, i_start=1)[1
][0]
    for j in range(len(predecessors)):
        if predecessors[j] == -9999:
            predecessors[j] = breadth_first_order(csgraph=graph, i_st
art=j+1)[1][j]

    print(names3[i])
    print("Distribution vector: ", dist_matrix)
    print("Predecessors: ", predecessors)
    print("Histograms for shortest path distributions and predecessor
s\n")
    plt.hist([dist_matrix, predecessors], bins = 'auto', label=['dis
t', 'pred'])
    plt.legend(loc='upper right')
    plt.show()
    print("------------------")
```

```
Five
Distribution vector:  [  0   1   2 ... 663 456 862]
Predecessors:  [  1   0   0 ... 530 574   2]
Histograms for shortest path distributions and predecessors
```



```
------------------
```

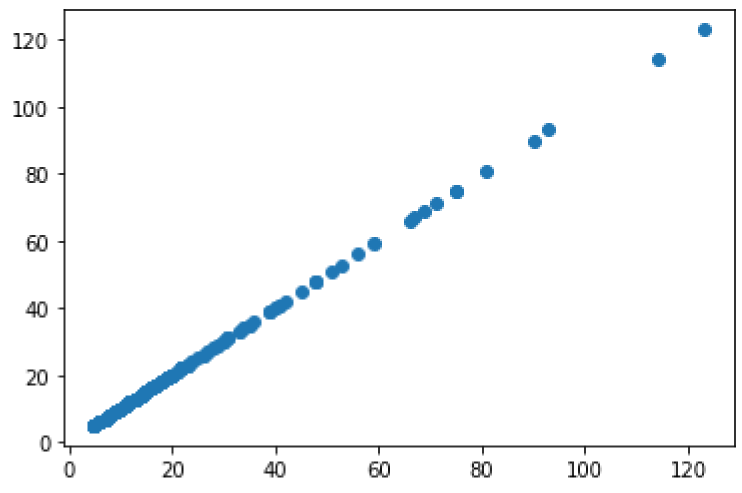
The skewing of degrees exaggerated even more with more nodes

In [97]:
```
eigenvalues(a_five)
```

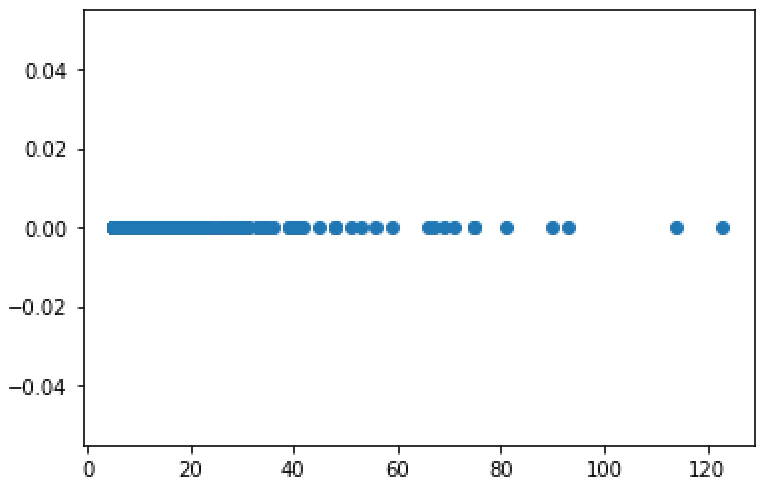
```
First 100 eigenvalues:
 [124.22887067 115.16068029  94.41170626  91.01481836  82.24168858
  77.15783924  75.47827567  72.75067796  69.36502789  67.91629187
  66.66481085  60.11651922  57.4090516   54.20049168  52.00809445
  49.42649727  48.98161201  48.90872014  45.9311437   43.78282396
  42.75579451  41.48405842  41.38665976  40.85246906  40.19212343
  39.98905024  39.72095697  37.21046704  36.83649917  35.50492699
  35.33282906  35.23439422  34.41468564  34.18764281  32.61496679
  32.35553226  32.19663452  31.73870394  31.17010135  30.96693402
  30.8393308   30.35424825  30.26647945  29.08846335  28.44123645
  28.14759022  28.26039002  26.62436503  26.54700912  26.10383611
  25.62498629  25.01416254  24.94401722  24.61159249  24.23582853
  24.15371713  23.97491169  23.57671387  23.22913071  23.08117118
  23.02461132  22.98327628  22.85164819  22.71486874  22.48093979
  22.43392112  22.10078718  22.08047149  21.74008494  21.58018326
  21.38779709  21.3415254   21.29068757  21.16607359  21.07046099
  20.99241076  20.84050532  20.81321449  20.58045199  20.48486304
  20.43684547  20.40299455  20.31369432  20.20398685  20.1260494
  20.09698878  19.90441796  19.83403782  19.81752823  19.46409177
  19.34928685  19.18409641  19.13620225  19.04332625  18.90017103
  18.80601259  18.52143205  18.35409761  18.19825412  18.11175009]
Spectral Gap, smallest non-zero eigenvalue for the first 100:  18.111
750087381605
```

In [98]:
```
scatter(a_five)
```

```
Pearson Correlation Coefficient:  1.0
```



In [99]:
```
d_c(a_five)
```



Overall, there are minor changes but we can see that as the algorithm runs, most of the patterns stay the same.