# ID2222 Data Mining

# Homework 4: Graph Spectra

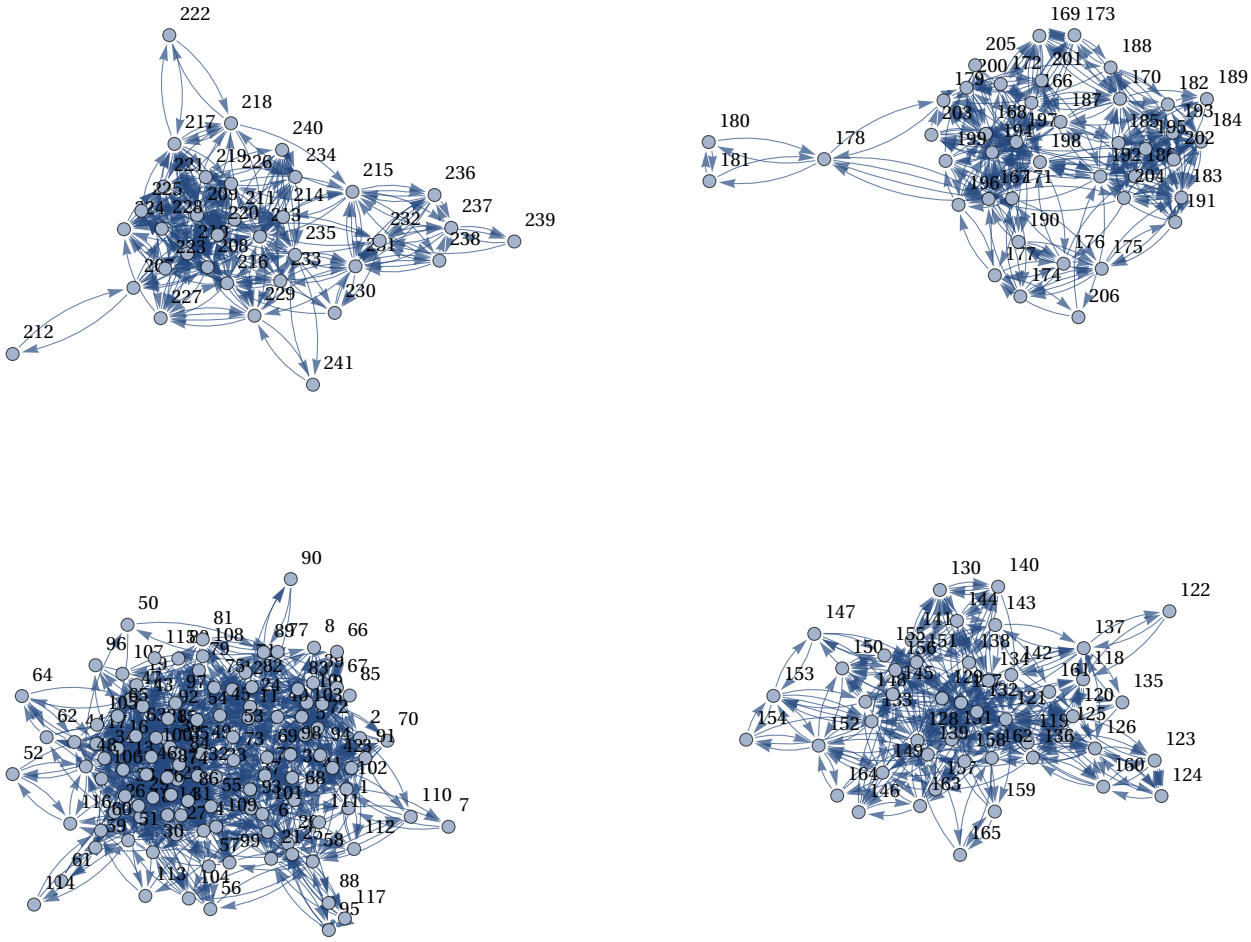
*Graph 1*

**Kim Hammar**
KTH Royal Institute of Technology

**Konstantin Sozinov**
KTH Royal Institute of Technology

## Graph Import

In[65]:=
```
SetDirectory[NotebookDirectory[]];
edgeList = Import["example1.csv","Data"];
graph = Graph[DirectedEdge@@@ edgeList,VertexLabels→"Name"];
```



# General Graph Properties

## Edge Count

In[68]:=
```
EdgeCount[graph];
```

2196

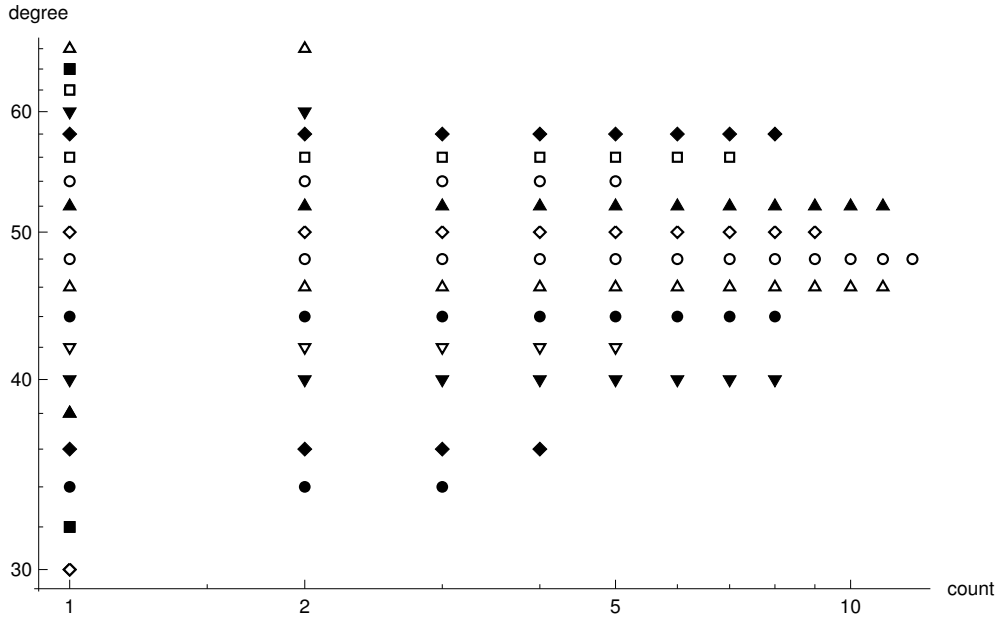
## Vertex Count

In[69]:=
```
VertexCount[graph];
```

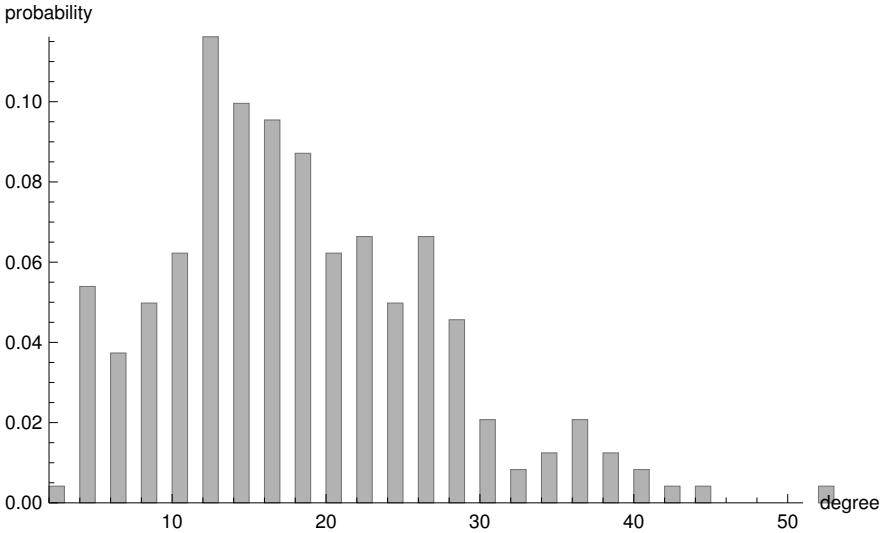
241

## Degree Distribution

In[70]:=
```
Histogram[VertexDegree[graph],{1},"Probability",AxesLabel→{"degree","probability"}];
ListLogLogPlot[GroupBy[VertexDegree[graph], Count], AxesLabel→{"count","degree"}];
```




Log-Log plot over degree distribution to do a rough test for powerlaw. Since the distribution is not linear it is probably not a powerlaw.




Regular Histogram plot over degree distribution.

## Global Clustering Coefficient

In[72]:=
```
GlobalClusteringCoefficient[graph];
```

$$\frac{1008}{4013}$$

# Graph Communities

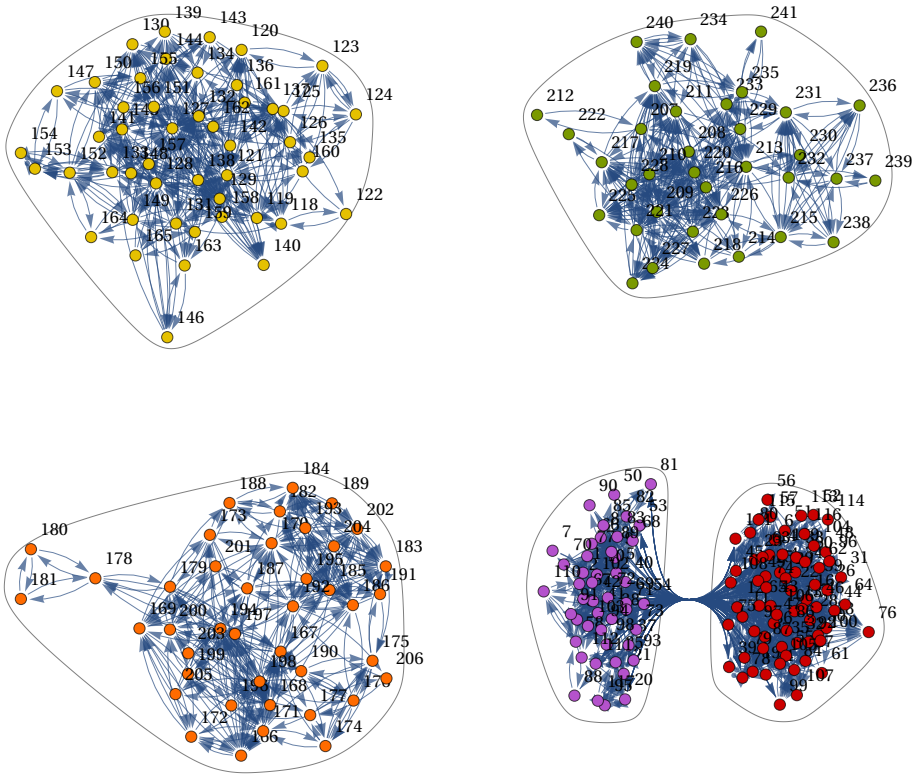
## Communities Count

In[73]:=
```
Length[FindGraphCommunities[graph]];
```

5

## Communities Plot

In[74]:=
```
CommunityGraphPlot[graph];
```



# Graph Spectra

# Graph Spectra

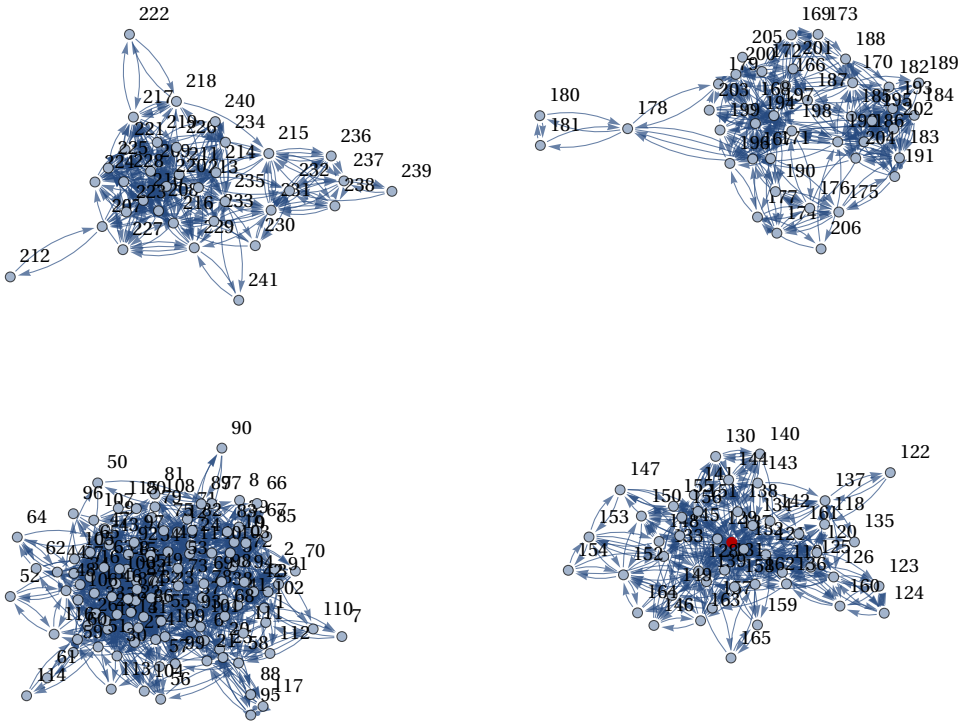
In[75]:=
```
A = AdjacencyMatrix[graph];
{eigenVals,eigenVecs} =Eigensystem[N[A]];
```

# Node Centralities

## PageRank Centrality

In[77]:=
```
MaxPageRankCentralNode = VertexList[graph][[Position[PageRankCentrality[graph],
Max[PageRankCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxPageRankCentralNode];
```

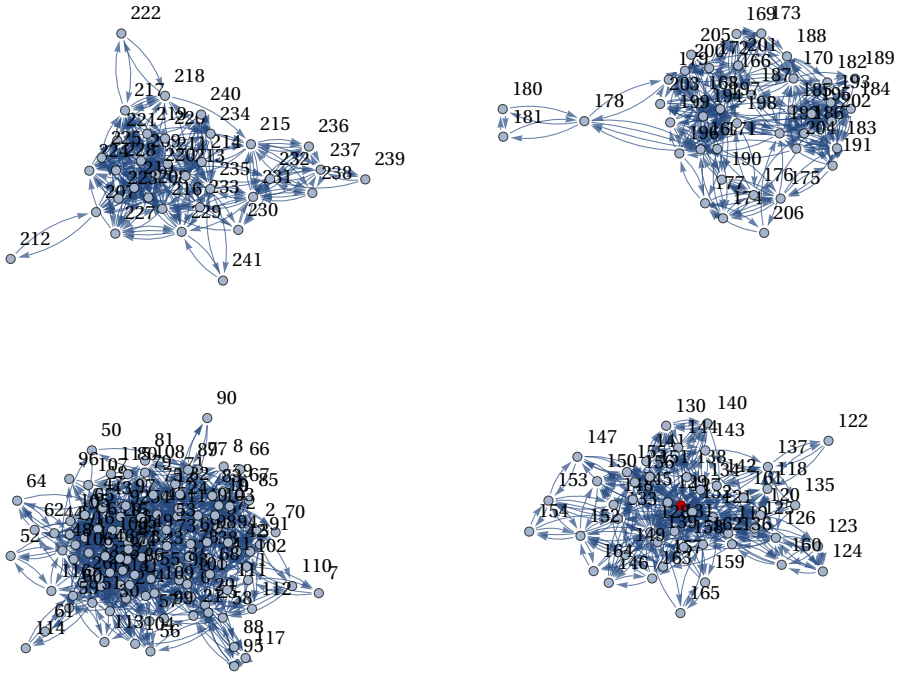
{127}



## Degree Centrality

In[79]:=
```
MaxDegreeCentralNode = VertexList[graph][[Position[DegreeCentrality[graph],
Max[DegreeCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxDegreeCentralNode];
```

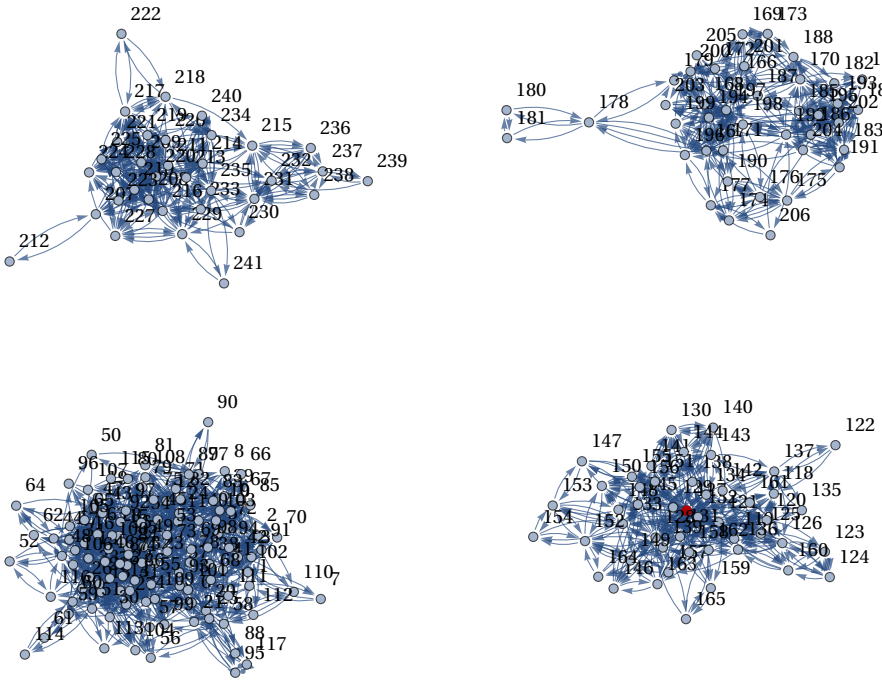
{127}



## Closeness Centrality

In[81]:=
```
MaxClosenessCentralityNode = VertexList[graph][[Position[ClosenessCentrality[graph],
Max[ClosenessCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxClosenessCentralityNode];
```

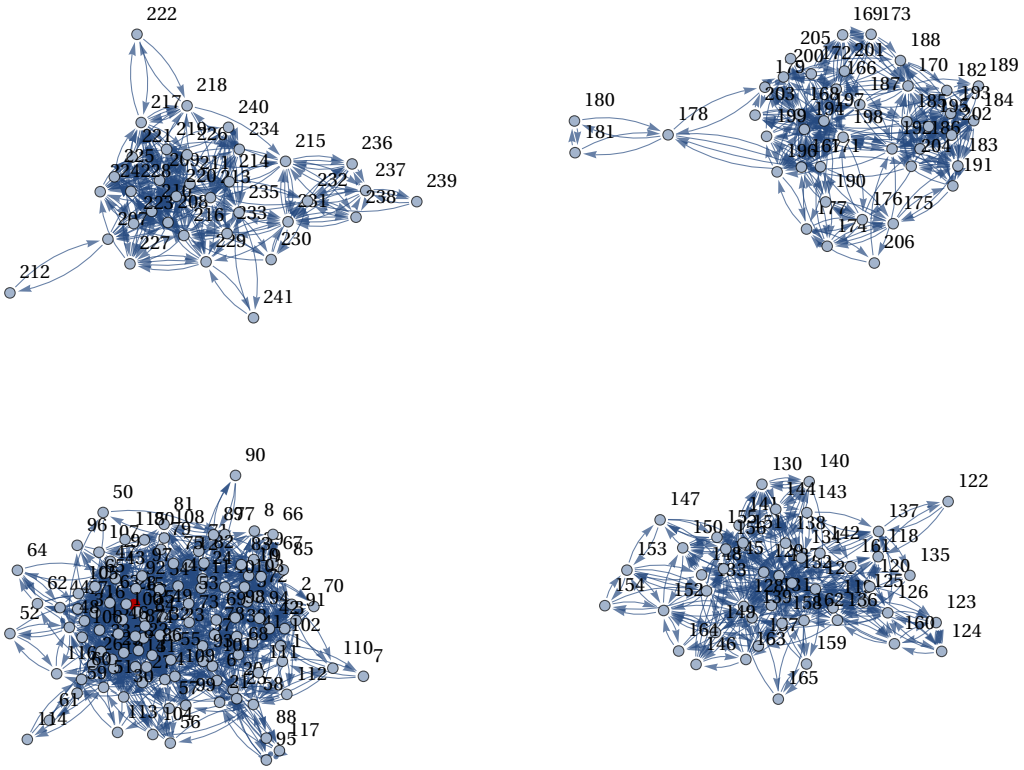
{127}



## Betweenness Centrality

In[83]:=
```
MaxBetweenessCentralityNode = VertexList[graph][[Position[BetweennessCentrality[graph],
Max[BetweennessCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxBetweenessCentralityNode];
```

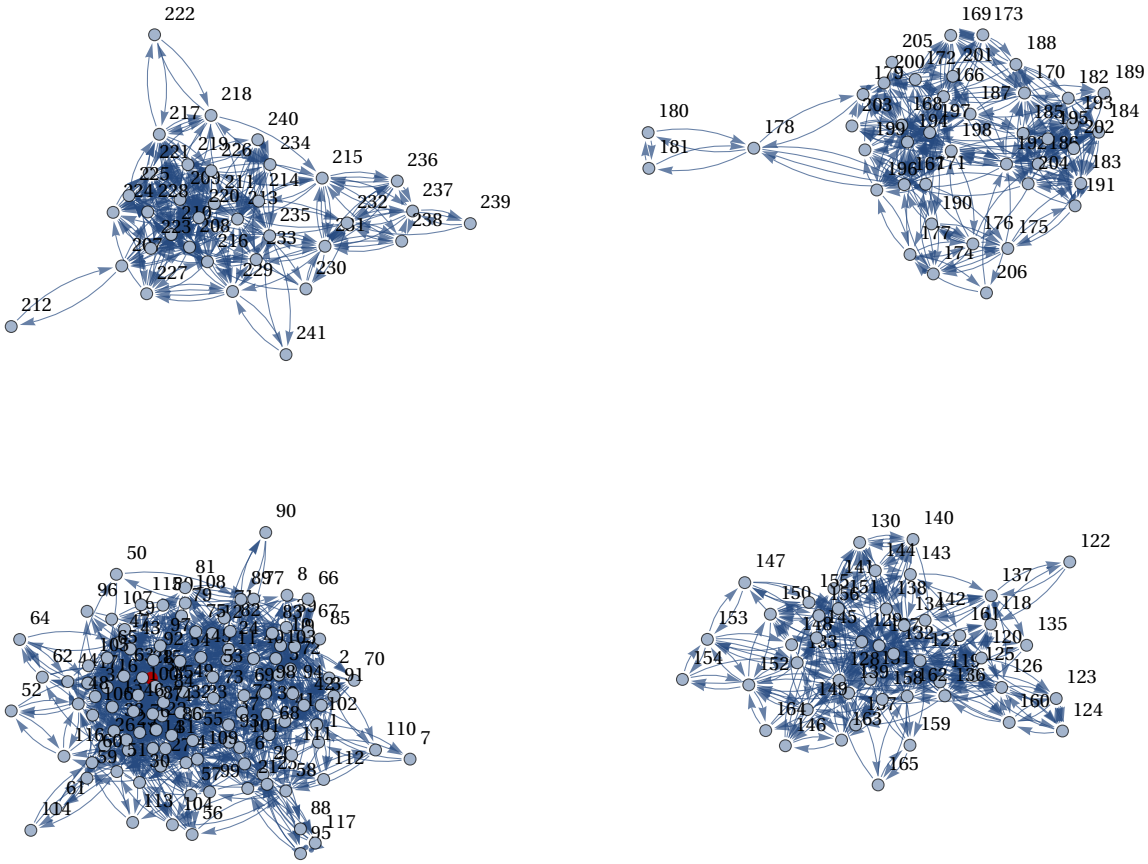
{15}



## EigenVector Centrality

In[85]:=
```
MaxEigenVectorCentralityNode = VertexList[graph][[Position[EigenvectorCentrality[graph]
Max[EigenvectorCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxEigenVectorCentralityNode];
```

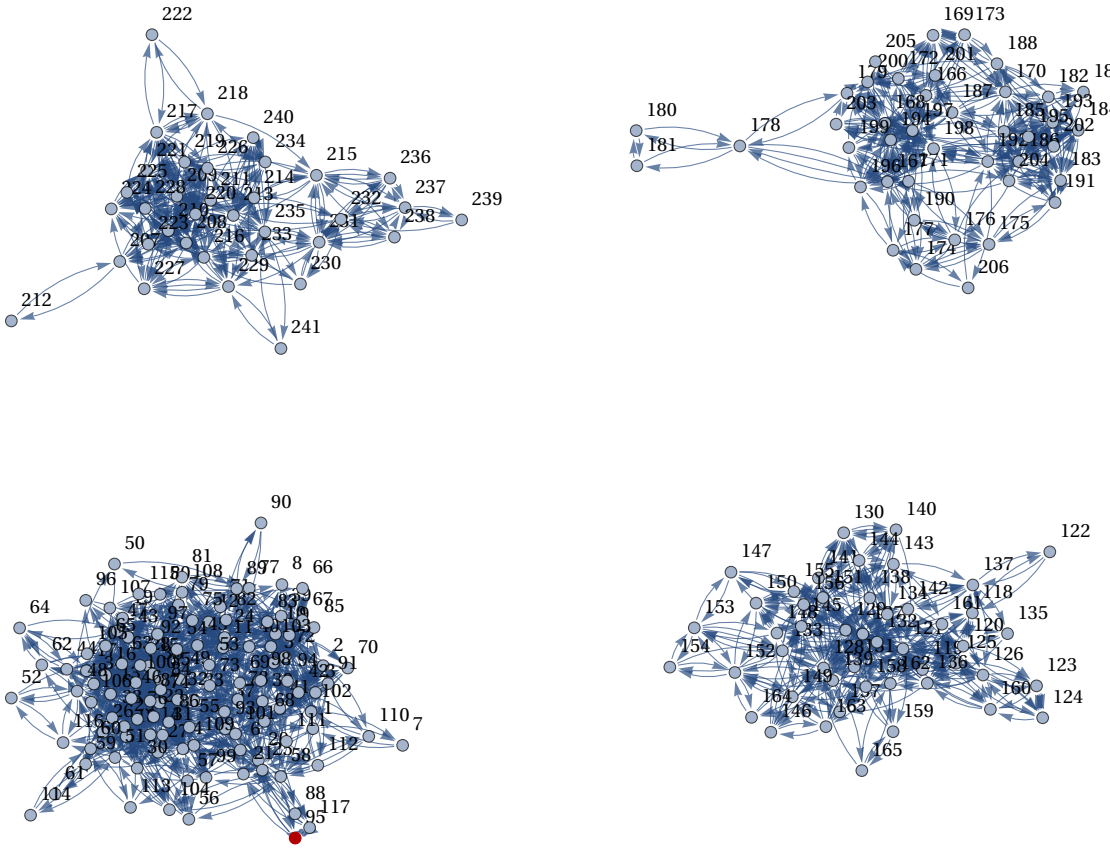
{15}



## Clustering Coefficient

In[87]:=
```
MaxClusterNode = VertexList[graph][[Position[LocalClusteringCoefficient[graph],
Max[LocalClusteringCoefficient[graph]]][[1]]]];
HighlightGraph[graph, MaxClusterNode];
```

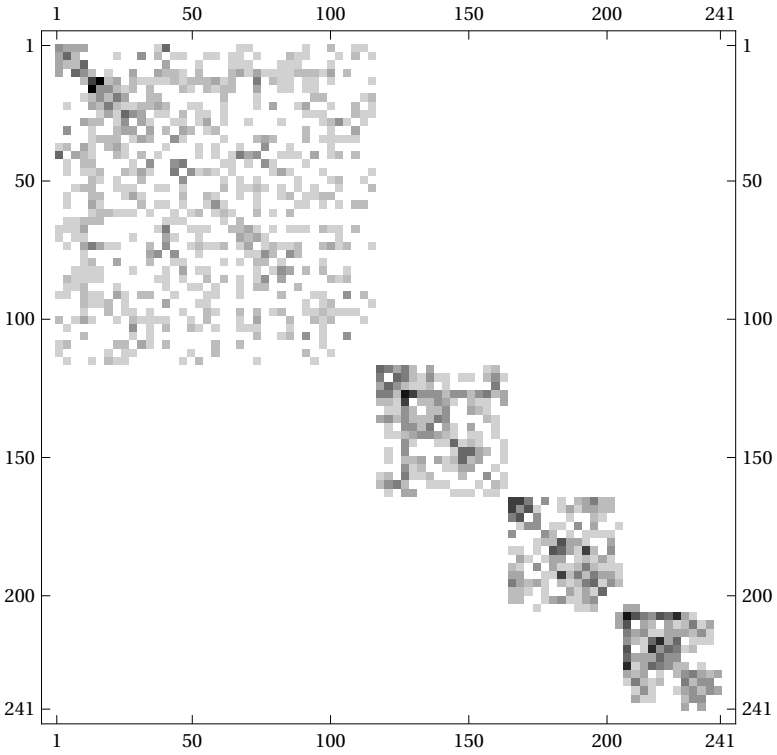
{95}



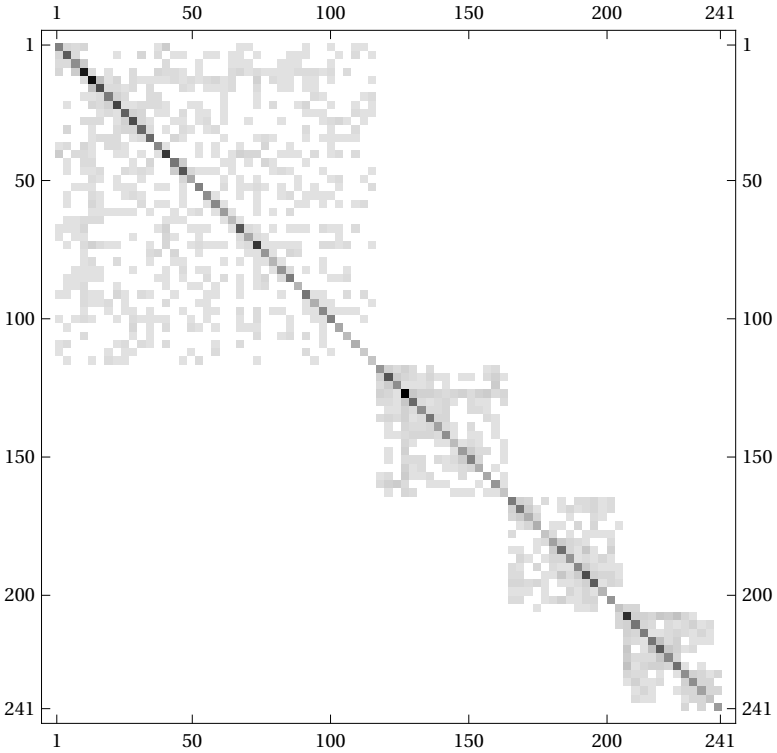
## Adjacency Matrix Heatmap, Non-Normalized Laplacian (Kirchoff), Affinity Matrix

Plotting the heatmap of the adjacency matrix is a common way to visualize a network, it is especially effective when the node partitons are ordered based on node-ids. In our case the partitions are perfectly ordered sequentially on the node ids so the heatmap gives a good indication of the number of clusters.
There exists multiple versions of the Laplacian matrix with small modifications, the Kirchoff matrix is one of them. The affinity matrix is computed as the Jordan Decomposition of the Adjaceny matrix. There are many spectrums that are useful for graph analysis, including the spectrum of the adjacency matrix, the transition matrix and the Laplacian matrix. However, the Laplacian matrix is the most useful for reasoning about the connectivity of the graph as well as its clustering.

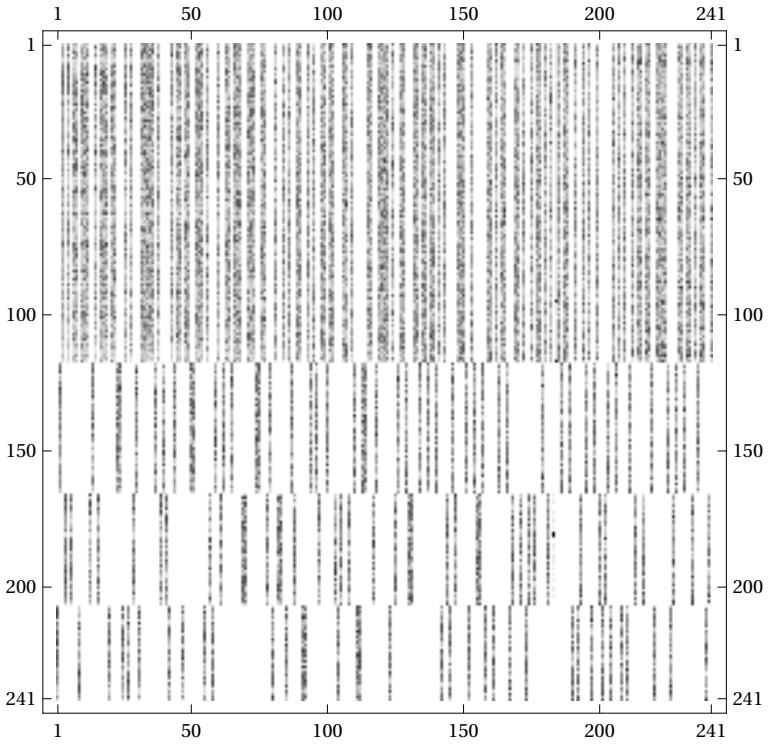
In[89]:=
```
MatrixPlot[A];
kirchoffLaplacianMatrix = KirchhoffMatrix[graph];
MatrixPlot[kirchoffLaplacianMatrix];
{affinityMatrix, s} = JordanDecomposition[N[Transpose[A]]];
MatrixPlot[affinityMatrix];
```




Adjacency Matrix plot




KirchoffLaplacian Matrix (not normalized)

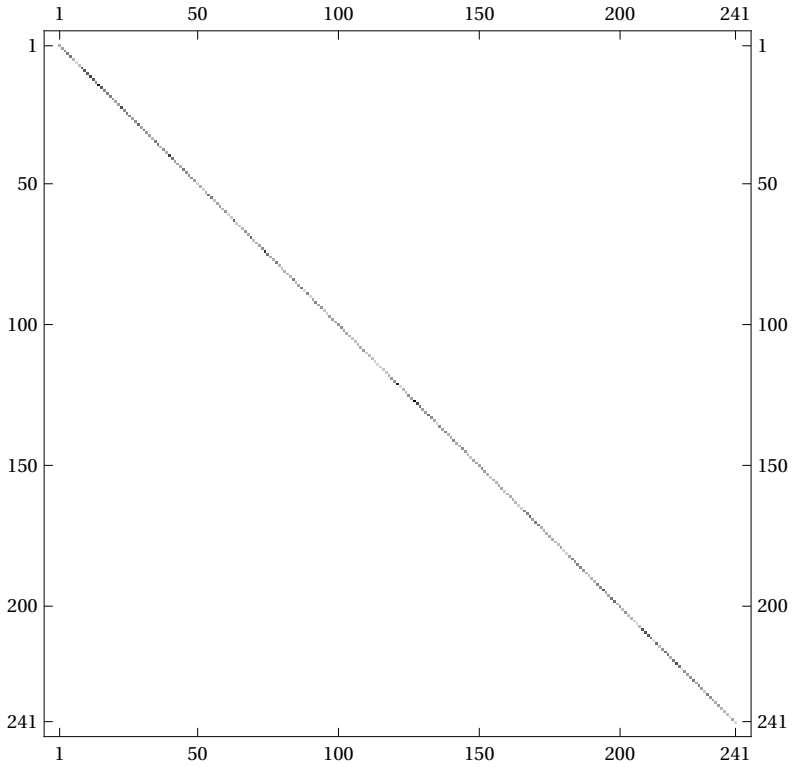



Affinity Matrix Plot

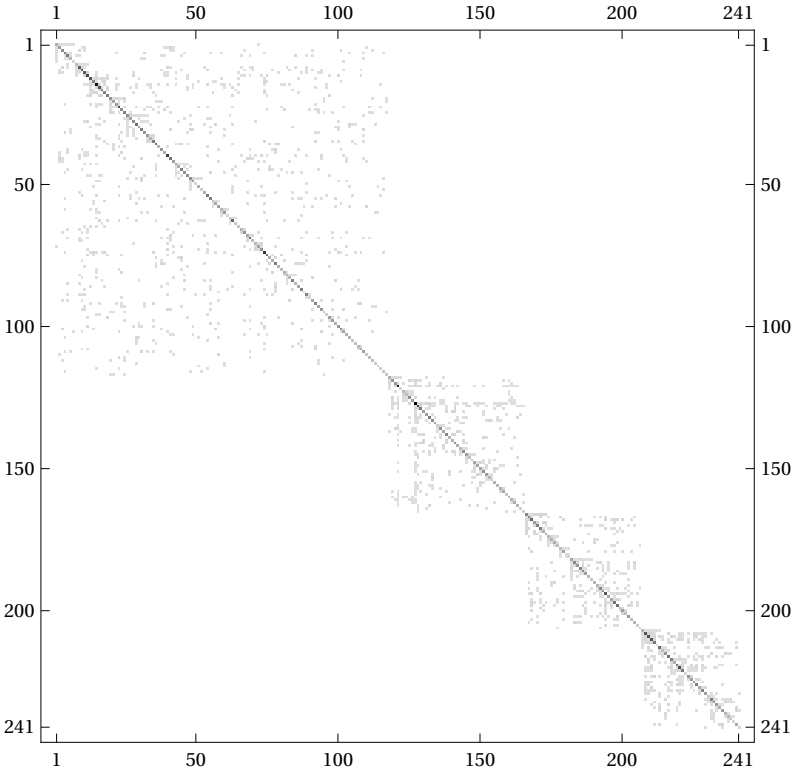
## Degree Matrix, Simple Laplacian

The Degree Matrix D is a diagonal matrix with the degree of each node i on the diagonal (i,i). Here I also compute the simple Laplacian matrix which is L = D - A.

In[94]:=
```
{n,n} = Dimensions[A];
DegreeMatrix = ConstantArray[0, {n,n}];
For[i = 1, i <= n, i++, DegreeMatrix[[i,i]] = Total[A[[i]]]];
MatrixPlot[DegreeMatrix];
simpleLaplacian = DegreeMatrix - A;
MatrixPlot[simpleLaplacian];
```




Degree Matrix Plot



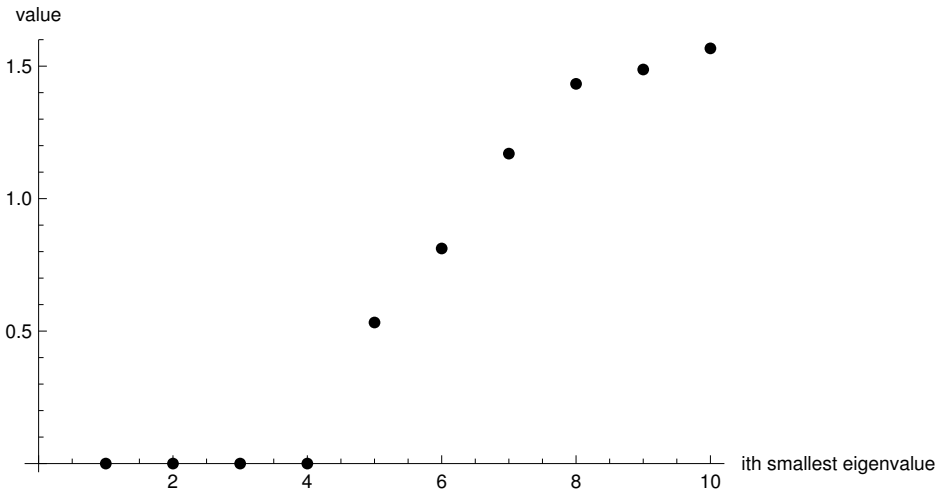

Simple Laplacian Matrix Plot

## EigenGap of Simple Laplacian and Fiedler Vector

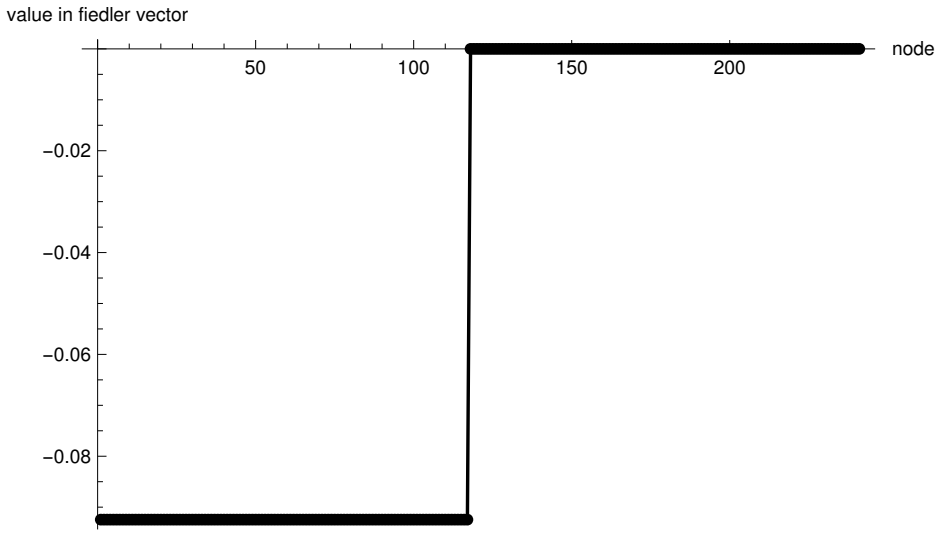
In the Laplacian we know that $\lambda_1 = 0$ with a corresponding eigenvector $v_1 = [1, \ldots, 1]$. This follows from the fact that the rows and the columns of the Laplacian sum up to 0 (each row contains number of -1 as number of neighbors plus one entry on the row which is the degree of the node). Further more we can tell by analyzing the higher-order eigenvalues how many connected components the graph has and whether it is a good expander or not. In our case $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0$ which means that the graph has 4 connected components. We also know from spectral graph theory that $\lambda_{n \leq 2}$which is in concordance with our results below. Furthermore, by looking at the eigen-gap between $\lambda_4$and $\lambda_5$we can tell whether it is close or not that there is a fifth disconnected component, and we can see that the fifth components seems to be quite well connected since the eigen-gap is quite large.

Finally, the eigenvector associated with $\lambda_2$, also know as the "Fiedler Vector", can give a bi-partition of the graph. The Fiedler does not indicate the k-clusters but it can indicate the optimal 2-clusters (and if applied recursively it can even find k clusters, but typically to exploit higher order eigenvectors is a better approach).

In[179]:=
```
{smallestEigenVals, smallestEigenVecs} = Eigensystem[N[simpleLaplacian], -10];
ListPlot[Reverse[Chop[smallestEigenVals]], AxesLabel→{"ith smallest eigenvalue","value"
fiedlerVector = smallestEigenVecs[[-2]];
ListLinePlot[Sort[fiedlerVector], AxesLabel→{"node","value in fiedler vector"}];
```




We can see that there is a gap between the 4th smallest eigenvalue and the 5th smallest of the simple Laplacian. This indicates that there are 4 clusters.

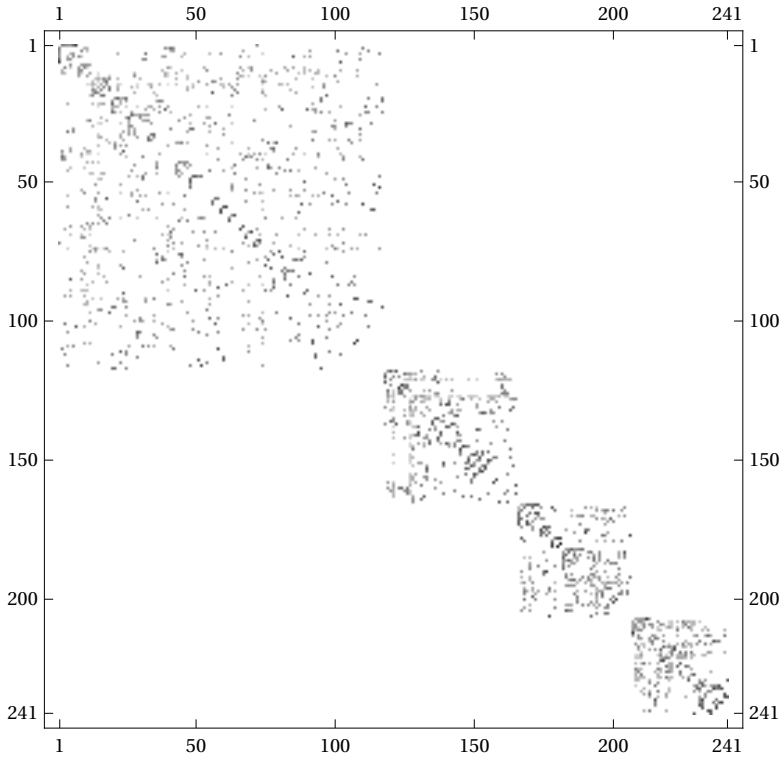



Sorted Fiedler Vector plot, gives a near optimal 2-way partitioning by assigning nodes to partitions based on their index in the Fiedler vector and the sign of the value in the vector.

## Ng, Jordan, & Weiss Laplacian

As mentioned, there are many variants of the laplacian matrix used in different contexts, they all have the same basic characteristics and differ only slightly. In the code snippet below the Laplacian of the Ng, Jordan & Weiss paper is computed as L = $D^{-1/2}\,\mathrm{AD}^{-1/2}$

In[104]:=
```
k = 4;
{n,n} = Dimensions[A];
njwLaplacian = MatrixPower[DegreeMatrix,-(1/2)].(A.MatrixPower[DegreeMatrix, -(1/2)]);
MatrixPlot[njwLaplacian];
```



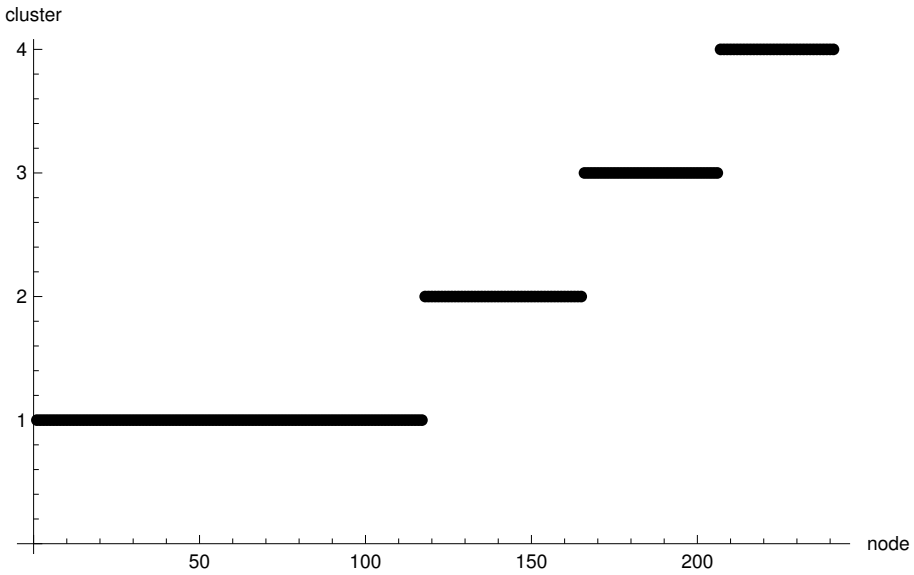

Ng, Jordan, & Weiss Laplacian

## Ng, Jordan, & Weiss Spectral Clustering

Here the bulk of the algorithm in the paper is implemented.

1. Decide k, we chose k = 4 since this was obvious from the preprocessing, e.g there are 4 connected components for instance.
2. The eigendecomposition of the Laplacian is computed
3. X is formed as a matrix with columns being the k largest eigenvectors
4. Define Y as X with all columns normalized to unit length
5. Cluster Y with K-means (a row in Y is considered as a datapoint to cluster)
6. Assign the original datapoints (from the adjacency matrix) to their clusters based on what their corresponding row in Y was clustered as.

In[108]:=
```
k = 4;
{largestEigenVals, largestEigenVecs} = Eigensystem[N[njwLaplacian],4];
X = Transpose[largestEigenVecs];
MatrixPlot[X];
{rows,cols} = Dimensions[X];
Y = ConstantArray[0, {rows,cols}];
For[i = 1, i <= cols, i++, Y[[All,i]] = Normalize[X[[All, i]]]];
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
```




As can be seen from the plot , the 4 clusters found by the spectral clustering are:

Cluster 1: Approximately 0-120

Cluster 2 : Approximately: 120-170

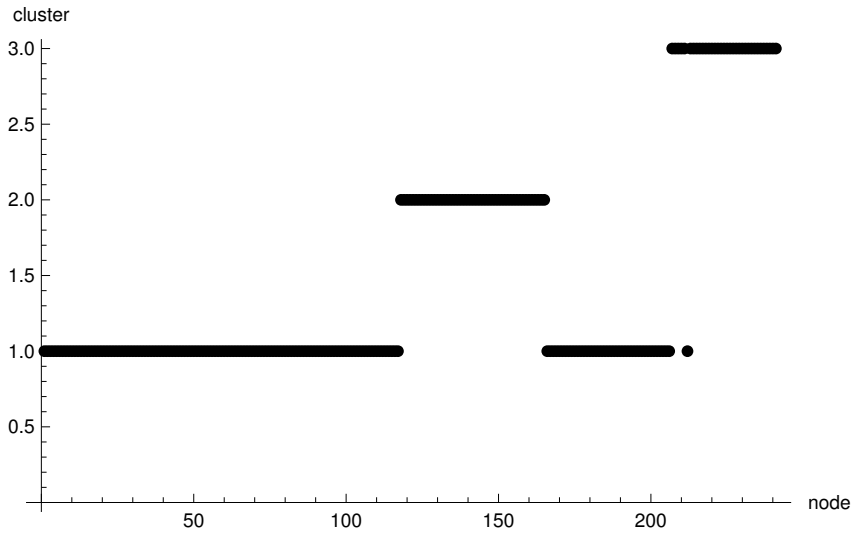
Cluster 3 : Approximately: 170-210

Cluster 4: Approximately: 210 -241

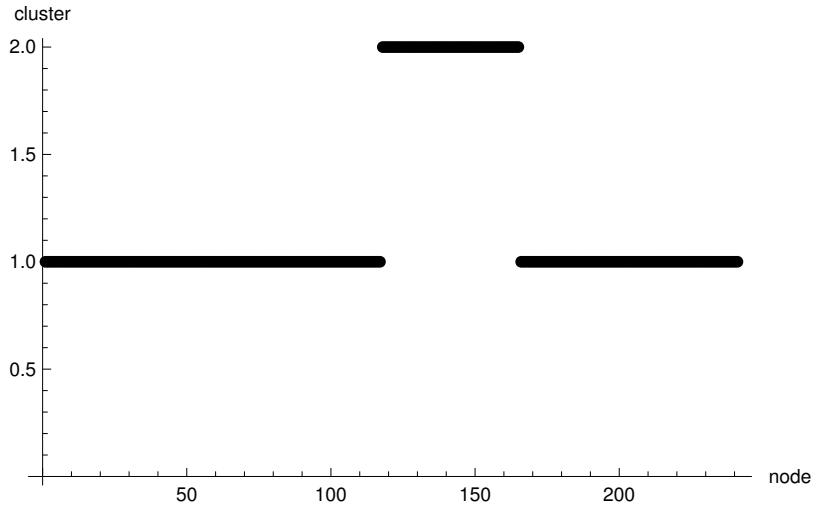
## Test clustering with "wrong" k

In[117]:=

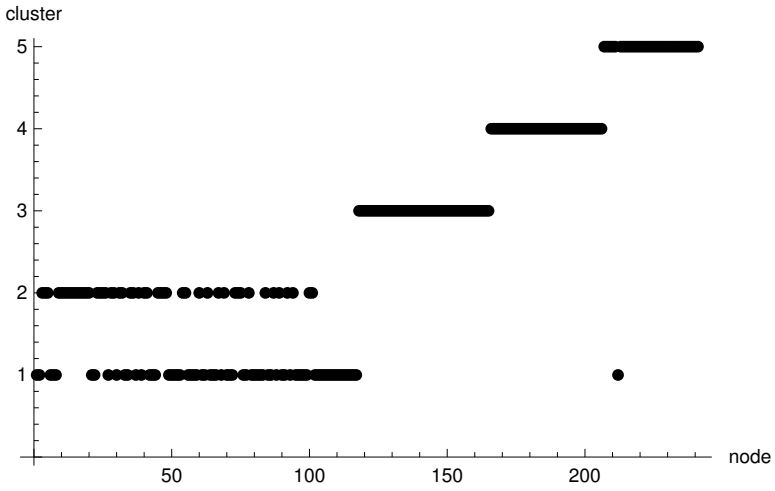
```
k = 3;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
k = 2;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
k = 5;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
k = 6;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
```


cluster
3.0
2.5
2.0
1.5
1.0
0.5
50
100
150
200
node

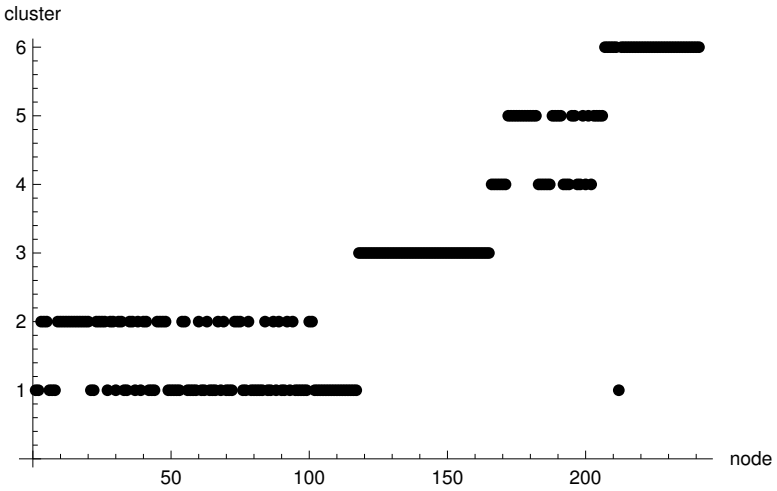
k = 3


cluster
2.0
1.5
1.0
0.5
50
100
150
200
node

k = 2


cluster
5
4
3
2
1
50
100
150
200
node

k = 5




k = 6

# ID2222 Data Mining

# Homework 4: Graph Spectra

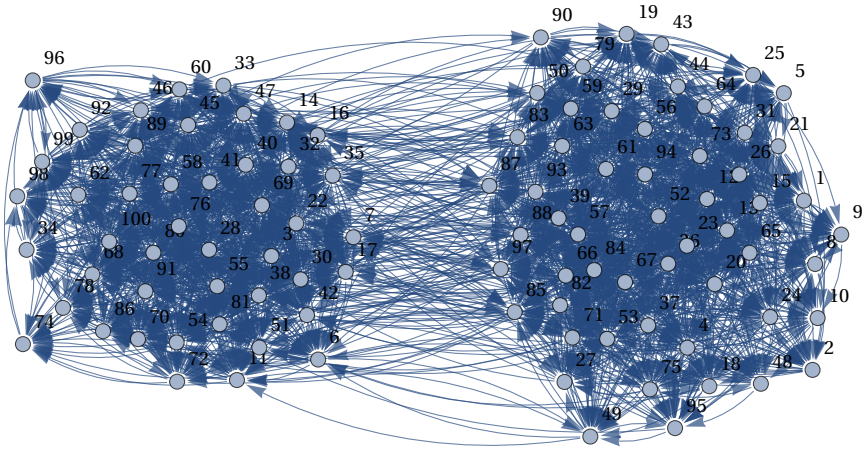
*Graph 2*

**Kim Hammar**
KTH Royal Institute of Technology

**Konstantin Sozinov**
KTH Royal Institute of Technology

## Graph Import

In[247]:=
```
SetDirectory[NotebookDirectory[]];
edgeList = Import["example2_clean.csv","Data"];
graph = Graph[DirectedEdge@@@ edgeList,VertexLabels→"Name"];
```



## General Graph Properties

### Edge Count

In[250]:=
```
EdgeCount[graph];
```

2418

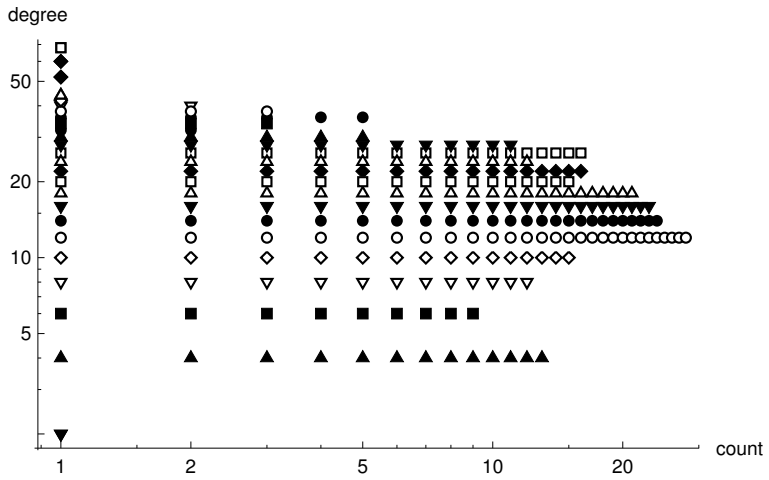
## Vertex Count

In[251]:=
```
VertexCount[graph];
```

100

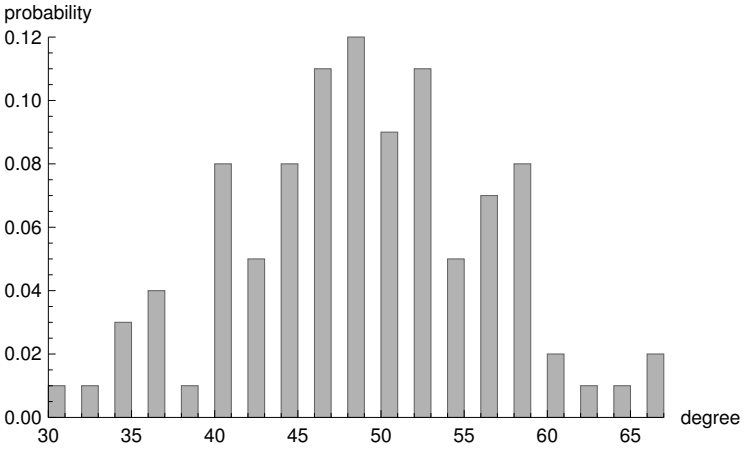
## Degree Distribution

In[252]:=
```
Histogram[VertexDegree[graph],{1},"Probability",AxesLabel→{"degree","probability"}];
ListLogLogPlot[GroupBy[VertexDegree[graph], Count], AxesLabel→{"count","degree"}];
```




Log-Log plot over degree distribution to do a rough test for powerlaw. The distribution has some linear tendency but not clear enough to be powerlaw.

In[254]:=




## Global Clustering Coefficient

In[255]:=
```
GlobalClusteringCoefficient[graph];
```

$$\frac{3649}{9580}$$

# Graph Communities

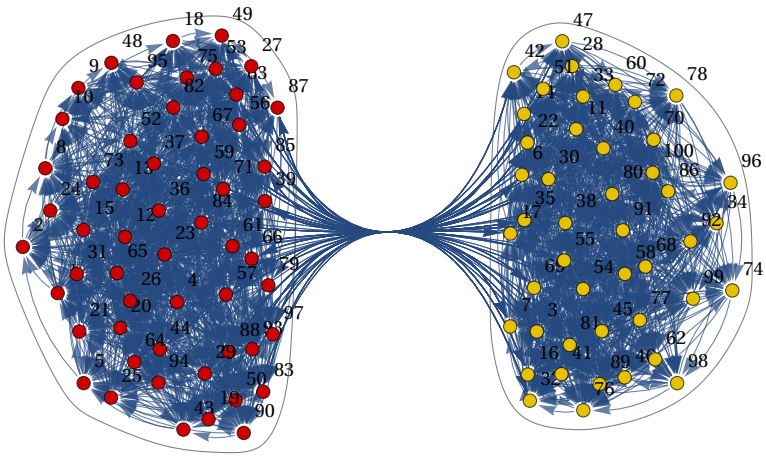
## Communities Count

In[256]:=
```
Length[FindGraphCommunities[graph]];
```

2

## Communities Plot

In[257]:=
```
CommunityGraphPlot[graph];
```



# Graph Spectra

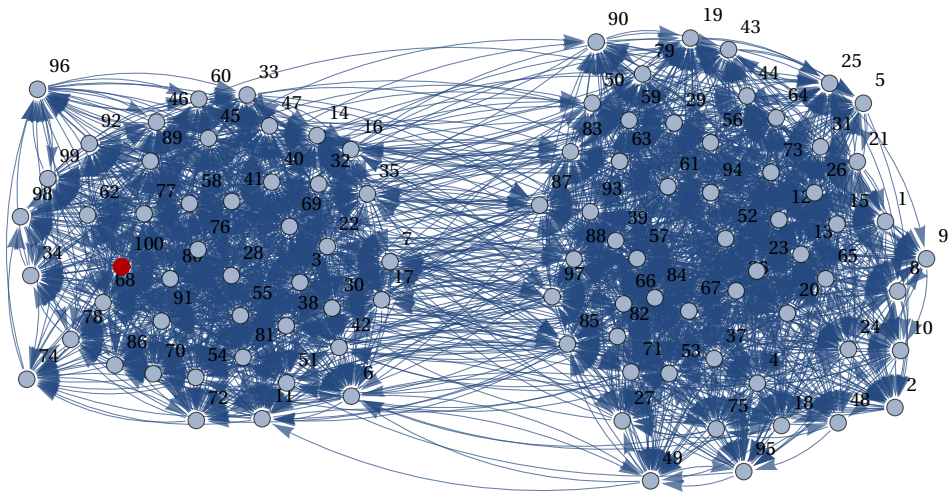
## Graph Spectra

In[258]:=
```
A = AdjacencyMatrix[graph];
{eigenVals,eigenVecs} =Eigensystem[N[A]];
```

# Node Centralities

## PageRank Centrality

In[260]:=
```
MaxPageRankCentralNode = VertexList[graph][[Position[PageRankCentrality[graph],
Max[PageRankCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxPageRankCentralNode];
```

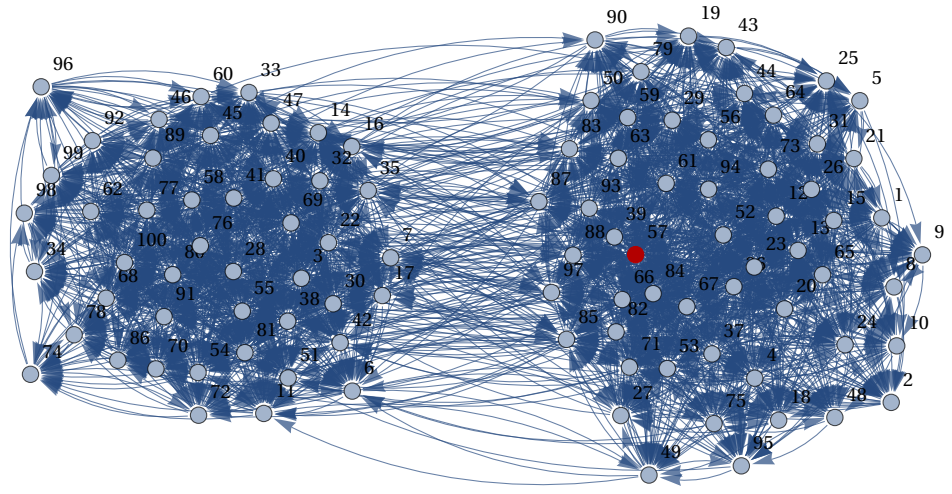


## Degree Centrality

In[262]:=

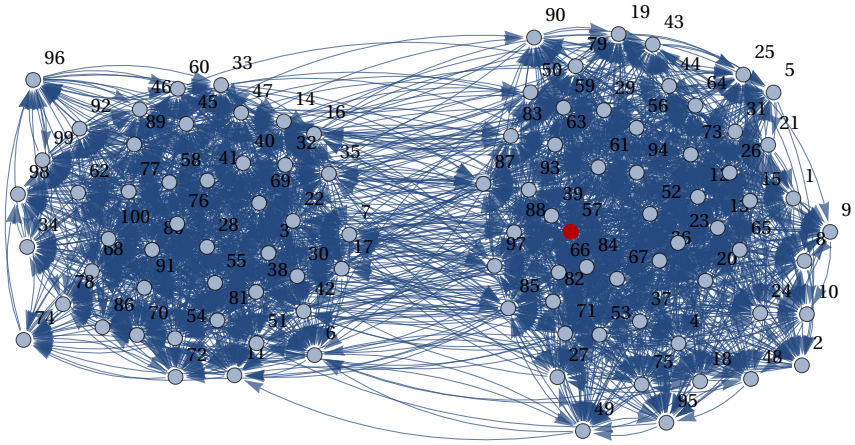
```
MaxDegreeCentralNode = VertexList[graph][[Position[DegreeCentrality[graph],
Max[DegreeCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxDegreeCentralNode];
```

In[264]:=



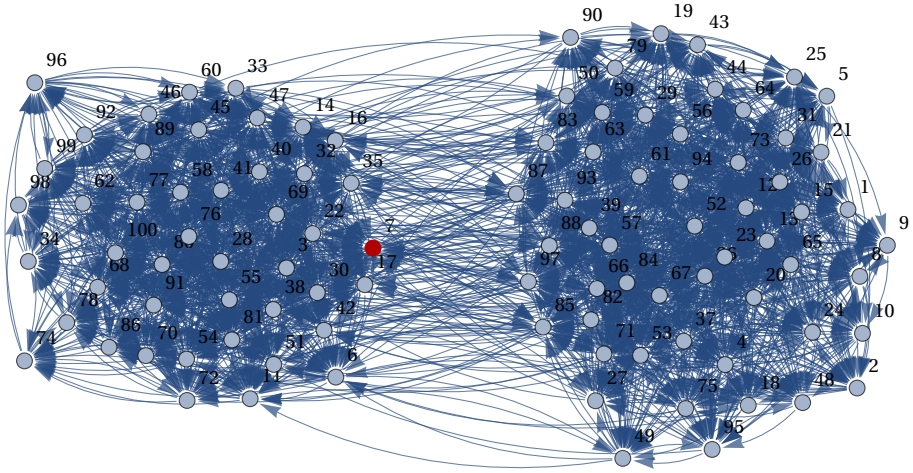
## Closeness Centrality

In[265]:=
```
MaxClosenessCentralityNode = VertexList[graph][[Position[ClosenessCentrality[graph],
Max[ClosenessCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxClosenessCentralityNode];
```



## Betweenness Centrality

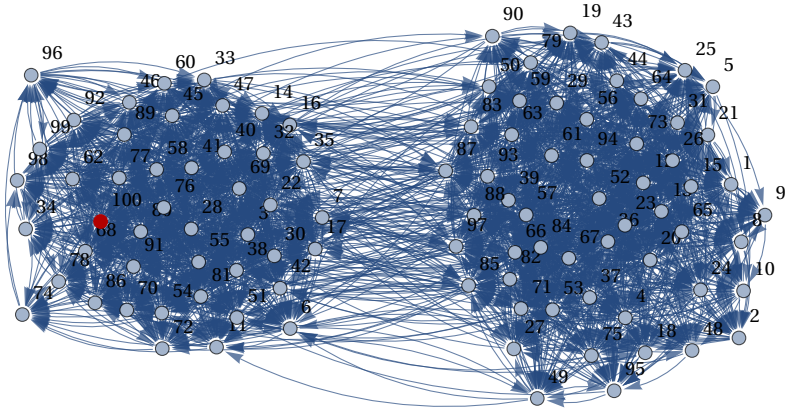
In[267]:=
```
MaxBetweenessCentralityNode = VertexList[graph][[Position[BetweennessCentrality[graph],
Max[BetweennessCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxBetweenessCentralityNode];
```



{15}

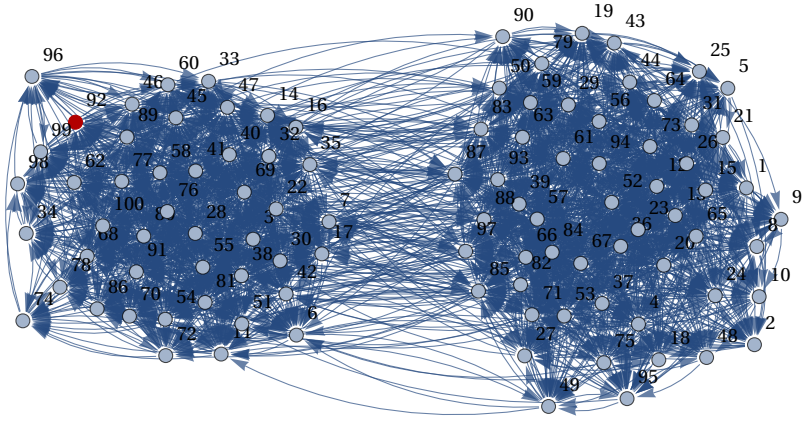
## EigenVector Centrality

In[269]:=
```
MaxEigenVectorCentralityNode = VertexList[graph][[Position[EigenvectorCentrality[graph]
Max[EigenvectorCentrality[graph]]][[1]]]];
HighlightGraph[graph, MaxEigenVectorCentralityNode];
```



## Clustering Coefficient

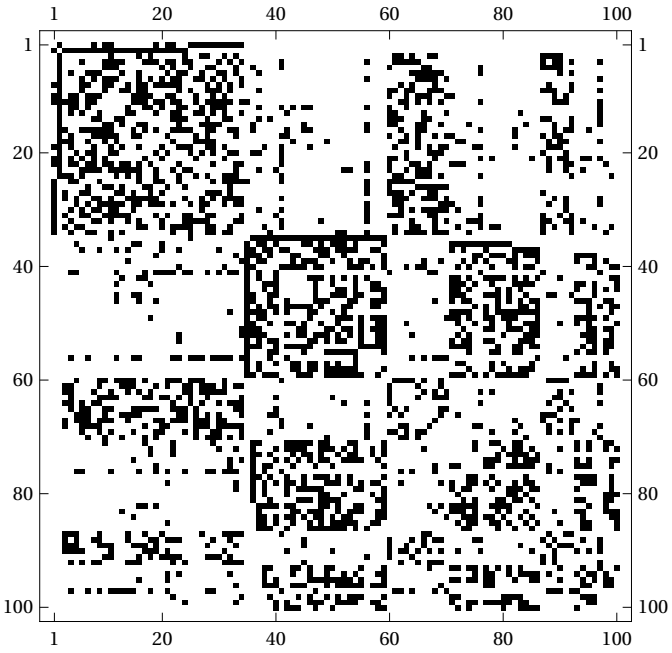
In[271]:=
```
MaxClusterNode = VertexList[graph][[Position[LocalClusteringCoefficient[graph],
Max[LocalClusteringCoefficient[graph]]][[1]]]];
HighlightGraph[graph, MaxClusterNode];
```



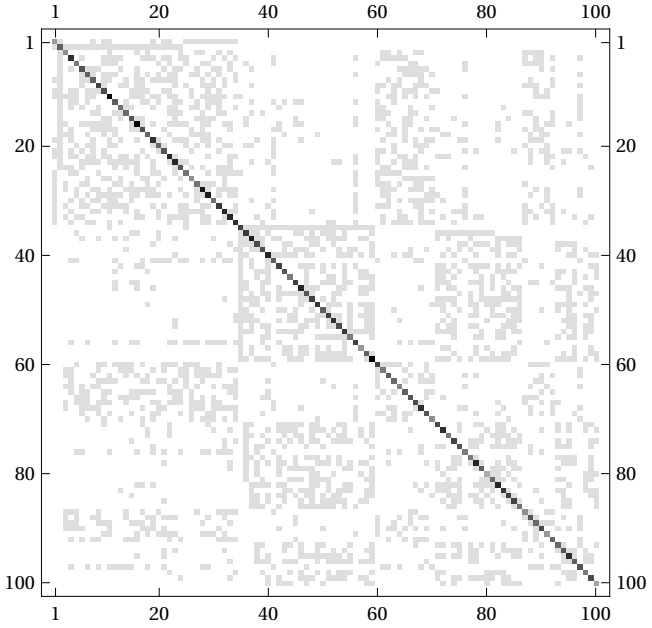
## Adjacency Matrix Heatmap, Non-Normalized Laplacian (Kirchoff), Affinity Matrix

Plotting the heatmap of the adjacency matrix is a common way to visualize a network, it is especially effective when the node partitons are ordered based on node-ids. In our case (unlike the first example graph) there is no correlation between node-ids and the partitions, thus the adjacency matrix is not as informative as it was for example graph 1
There exists multiple versions of the Laplacian matrix with small modifications, the Kirchoff matrix is one of them. The affinity matrix is computed as the Jordan Decomposition of the Adjaceny matrix. There are many spectrums that are useful for graph analysis, including the spectrum of the adjacency matrix, the transition matrix and the Laplacian matrix. However, the Laplacian matrix is the most useful for reasoning about the connectivity of the graph as well as its clustering.

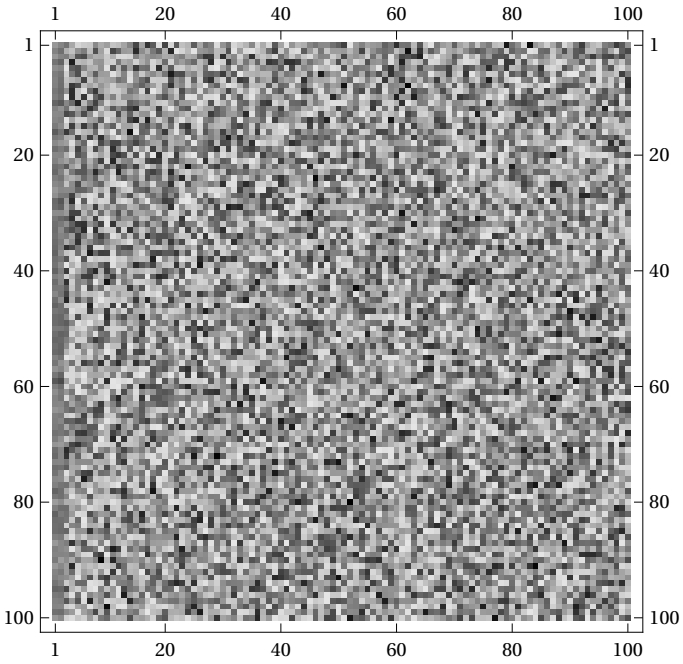
In[273]:=
```
MatrixPlot[A];
kirchoffLaplacianMatrix = KirchhoffMatrix[graph];
MatrixPlot[kirchoffLaplacianMatrix];
{affinityMatrix, s} = JordanDecomposition[N[Transpose[A]]];
MatrixPlot[affinityMatrix];
```




Adjacency Matrix plot




KirchoffLaplacian Matrix (not normalized)

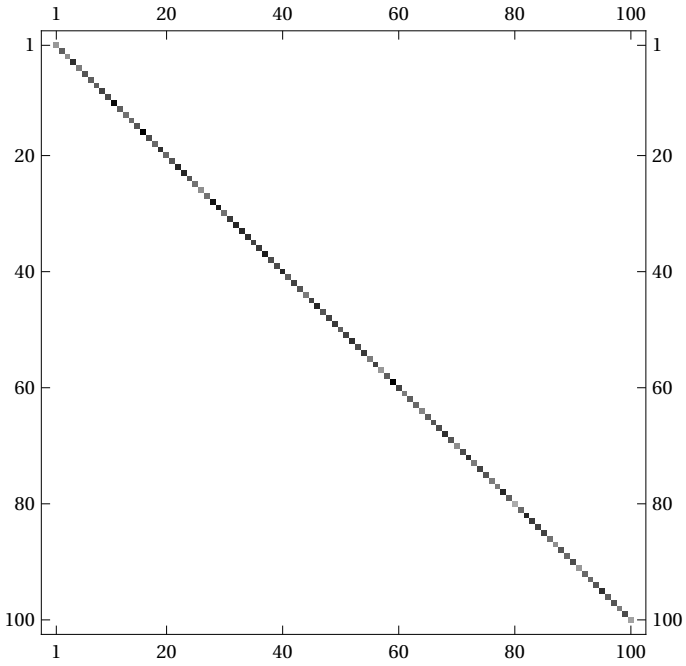



Affinity Matrix Plot

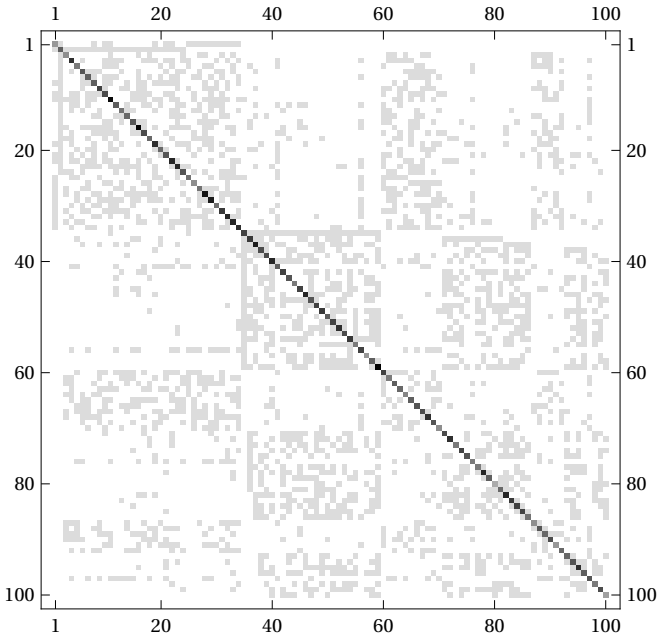
## Degree Matrix, Simple Laplacian

The Degree Matrix D is a diagonal matrix with the degree of each node i on the diagonal (i,i). Here I also compute the simple Laplacian matrix which is L = D - A.

In[278]:=
```
{n,n} = Dimensions[A];
DegreeMatrix = ConstantArray[0, {n,n}];
For[i = 1, i <= n, i++, DegreeMatrix[[i,i]] = Total[A[[i]]]];
MatrixPlot[DegreeMatrix];
simpleLaplacian = DegreeMatrix - A;
MatrixPlot[simpleLaplacian];
```




Degree Matrix Plot



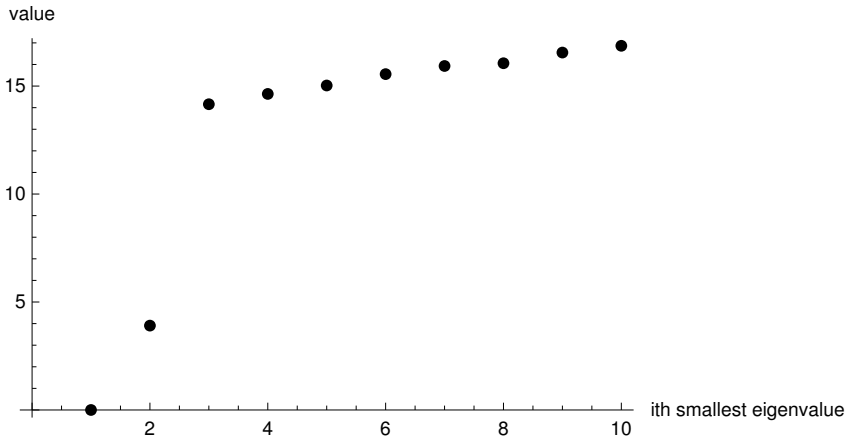

Simple Laplacian Matrix Plot

## EigenGap of Simple Laplacian and Fiedler Vector

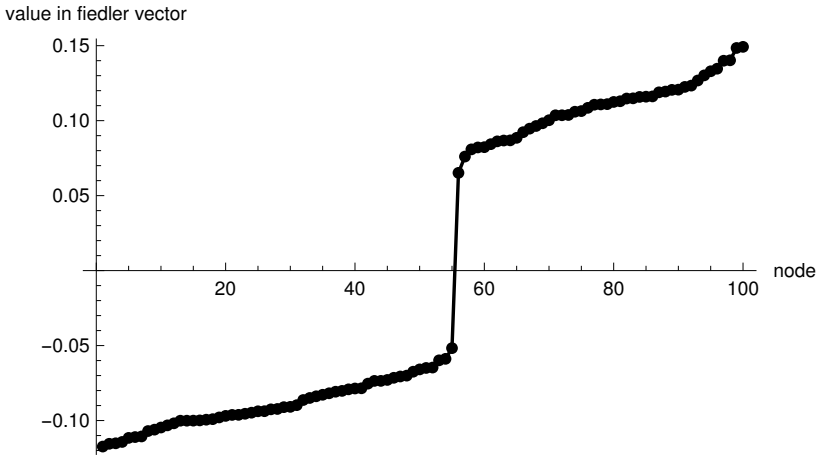
In the Laplacian we know that $\lambda_1 = 0$ with a corresponding eigenvector $v_1 = [1, \ldots, 1]$. This follows from the fact that the rows and the columns of the Laplacian sum up to 0 (each row contains number of -1 as number of neighbors plus one entry on the row which is the degree of the node). Further more we can tell by analyzing the higher-order eigenvalues how many connected components the graph has and whether it is a good expander or not. In our case $\lambda_1 = \lambda_2 = 0$ which means that the graph has 2 connected components. We also know from spectral graph theory that $\lambda_{n \leq 2}$which is in concordance with our results below. Furthermore, by looking at the eigen-gap between $\lambda_2$and $\lambda_3$we can tell whether it is close or not that there is a third disconnected component, and we can see that the third component seems to be quite well connected since the eigen-gap is quite large.

Finally, the eigenvector associated with $\lambda_2$, also know as the "Fiedler Vector", can give a bi-partition of the graph. The Fiedler does not indicate the k-clusters but it can indicate the optimal 2-clusters (and if applied recursively it can even find k clusters, but typically to exploit higher order eigenvectors is a better approach).

```
{smallestEigenVals, smallestEigenVecs} = Eigensystem[N[simpleLaplacian], -10];
ListPlot[Reverse[Chop[smallestEigenVals]], AxesLabel→{"ith smallest eigenvalue","value"
fiedlerVector = smallestEigenVecs[[-2]];
ListLinePlot[Sort[fiedlerVector], AxesLabel→{"node","value in fiedler vector"}];
```




We can see that there is a gap between the 2th smallest eigenvalue and the 3th smallest eigenvalue of the simple Laplacian. This indicates that there are 2 clusters.

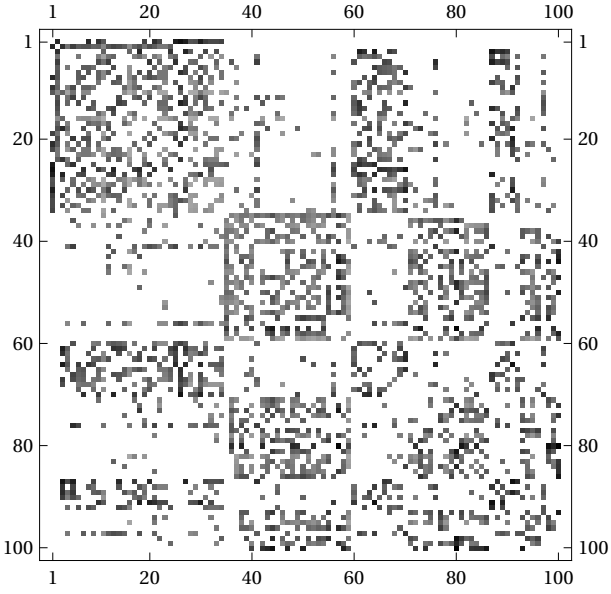



Sorted Fiedler Vector plot, gives a near optimal 2-way partitioning by assigning nodes to partitions based on their index in the Fiedler vector and the sign of the value in the vector.

## Ng, Jordan, & Weiss Laplacian

As mentioned, there are many variants of the laplacian matrix used in different contexts, they all have the same basic characteristics and differ only slightly. In the code snippet below the Laplacian of the Ng, Jordan & Weiss paper is computed as L = $D^{-1/2}\,\mathrm{AD}^{-1/2}$

In[288]:=
```
k = 2;
{n,n} = Dimensions[A];
njwLaplacian = MatrixPower[DegreeMatrix,-(1/2)].(A.MatrixPower[DegreeMatrix, -(1/2)]);
MatrixPlot[njwLaplacian];
```



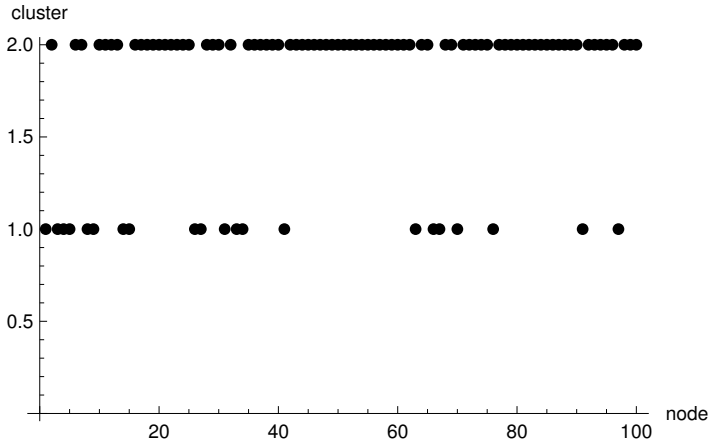

Ng, Jordan, & Weiss Laplacian

## Ng, Jordan, & Weiss Spectral Clustering

Here the bulk of the algorithm in the paper is implemented.

1. Decide k, we chose k = 2 since this was obvious from the preprocessing, e.g there are 2 connected components for instance.
2. The eigendecomposition of the Laplacian is computed
3. X is formed as a matrix with columns being the k largest eigenvectors
4. Define Y as X with all columns normalized to unit length
5. Cluster Y with K-means (a row in Y is considered as a datapoint to cluster)
6. Assign the original datapoints (from the adjacency matrix) to their clusters based on what their corresponding row in Y was clustered as.

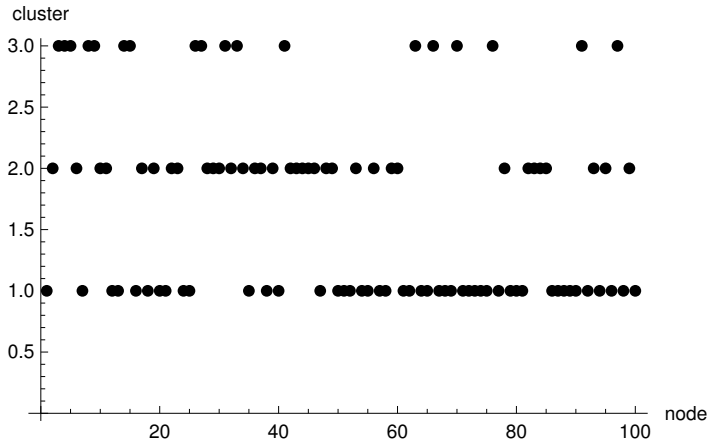
In[292]:=
```
k = 2;
{largestEigenVals, largestEigenVecs} = Eigensystem[N[njwLaplacian],4];
X = Transpose[largestEigenVecs];
MatrixPlot[X];
{rows,cols} = Dimensions[X];
Y = ConstantArray[0, {rows,cols}];
For[i = 1, i <= cols, i++, Y[[All,i]] = Normalize[X[[All, i]]]];
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
```



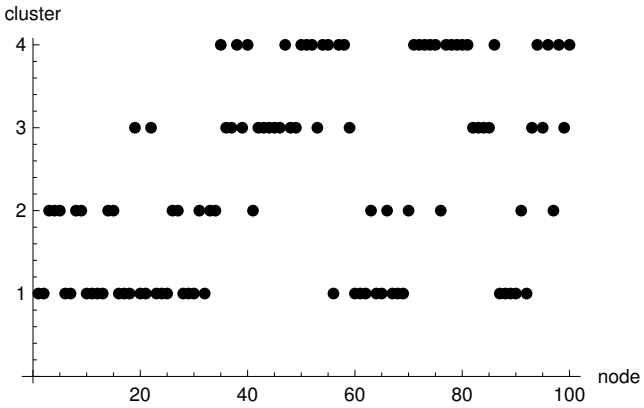

As can be seen from the plot , the 2 clusters have quite mixed node-ids. One cluster is larger than the other.

## Test clustering with "wrong" k

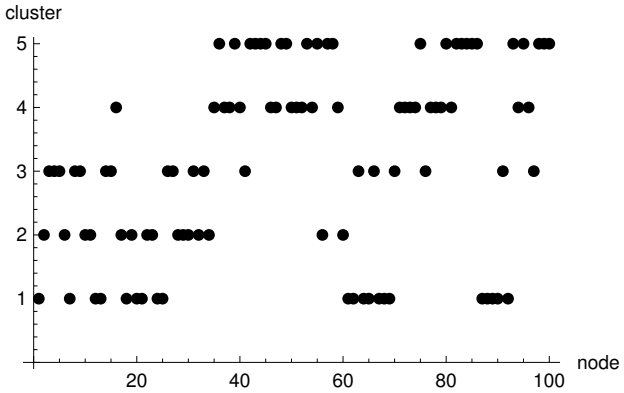
In[301]:=
```
k = 3;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
k = 4;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
k = 5;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
k = 6;
clusters = ClusteringComponents[Y,k,1, Method→ "KMeans"];
ListPlot[clusters, AxesLabel→{"node","cluster"}];
```



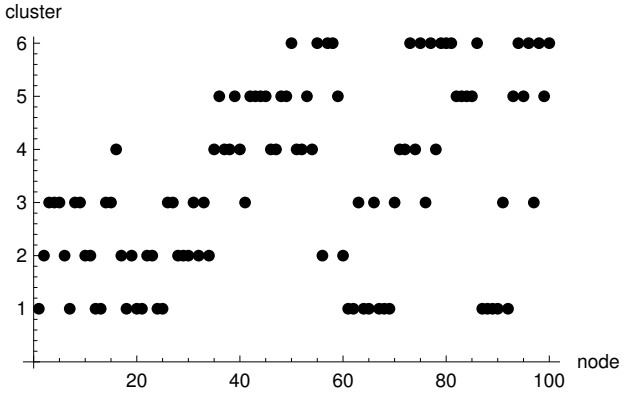

k = 3




k = 4


cluster
5
4
3
2
1
20
40
60
80
100
node

k = 5


cluster
6
5
4
3
2
1
20
40
60
80
100
node

k = 6