L10 Spectral Graph Bipartitioning

Ivan Slapničar

May 8, 2018

1 Spectral Graph Bipartitioning

Many data clustering problems can be interpreted as clustering of vertices of graphs. **Graph bipartitioning problem** is to partition vertices into subsets such that the connections within subsets are stronger than the connections between different subsets.

Partition of the vertices into two subsetts is done according to signs of the eigenvectors of the second smallest eigenvalue of the Laplacian matrix.

1.1 Prerequisites

The reader should be familiar with the basic graph theory, linear algebra and, in particular, eigenvalues and eigenvectors.

1.2 Competences

The reader should be able to apply graph spectral bipartitioning and recursive bipartitioning to data clustering problems.

Credits: The notebook is based on [Mir05].

1.3 Graphs

For more details, see [Hae14] and [BC14] and the references therein.

References

[Mir05] I. Mirošević, 'Spectral Graph Partitioning and Application to Knowledge Extraction', M.Sc. Thesis, University of Zagreb, 2005 (in Croatian).

[Hae14] W. H. Haemers, Matrices and Graphs, in L. Hogben, ed., 'Handbook of Linear Algebra', pp. 39.1-39.14, CRC Press, Boca Raton, 2014.

[BC14] S. Butler and F. Chung, Spectral Graph Theory, in L. Hogben, ed., 'Handbook of Linear Algebra', pp. 47.1-47.6, CRC Press, Boca Raton, 2014.

[HK04] D. J. Higham and M. Kibble, A Unified View of Spectral Clustering, Mathematics Research Report 2, University of Strathclyde, 2004.

1.3.1 Definitions

A **weighted graph** is an ordered triplet $G = (V, E, \omega)$, where $V = \{1, 2, 3, ..., n\}$ is the set of **vertices** , $E = \{(i, j)\}$ is a set of **edges** connecting vertices, and ω is a set of **weights** of edges. We assume G is undirected.

An **adjacency matrix** of graph G is the matrix A defined as $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{otherwise} \end{cases}$.

A **weight matrix** of graph G is the matrix W defined as $W_{ij} = \begin{cases} \omega(e) & \text{if } e = (i,j) \in E, \\ 0 & \text{otherwise} \end{cases}$.

A **Laplacian matrix** of graph G is the matrix $L = D - W$, where $D = \text{diag}(d_1, d_2, \ldots, d_n)$ with $d_i = \sum_{k=1}^{n} W_{ik}$ for $i = 1, \ldots, n$.

A **normalized Laplacian matrix** is the matrix $L_n = D^{-1/2} L D^{-1/2} \equiv D^{-1/2}(D - W)D^{-1/2}$ (*the scaled matrix of* L).

An **incidence matrix** of graph G is the $|V| \times |E|$ matrix I_G. Each row of $I : G$ corresponds to a vertex of G and each column corresponds to an edge of G. In the column corresponding to en edge $e = (i, j)$, all elements are zero except the ones in the i-th and j-th row, which are equal to $\sqrt{\omega(e)}$ and $-\sqrt{\omega(e)}$, respectively.

1.3.2 Examples

Graph types and algorithms are implemented in the package LightGraphs.jl.

Plotting graphs is done by the packages GraphPlot.jl.

As a small inconvenience, we can only plot unweighted graphs and plot weights as node labels.

```
In [1]: using LightGraphs
        using GraphPlot

In [2]: # whos(LightGraphs)

In [3]: # Sources, targets and weights
        n=7
        sn=[1,1,1,2,2,3,3,3,5,5,6]
        tn=[2,3,4,4,5,4,6,7,6,7,7]
        wn=[2,3,4,7,1,3,2,1,7,3,5]
        [sn tn wn]

Out[3]: 11×3 Array{Int64,2}:
         1  2  2
         1  3  3
         1  4  4
```

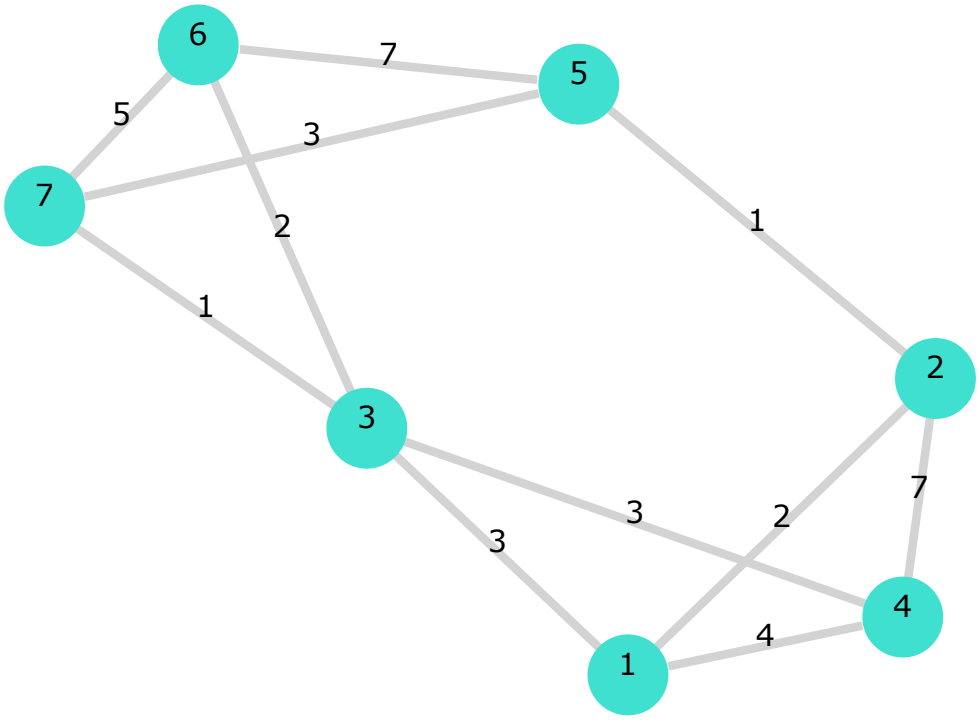
```
 2  4  7
 2  5  1
 3  4  3
 3  6  2
 3  7  1
 5  6  7
 5  7  3
 6  7  5
```

In [4]:
```
# Create the graph
G=Graph(n)
for i=1:length(sn)
    add_edge!(G,sn[i],tn[i])
end
G
```

Out[4]:
```
{7, 11} undirected simple Int64 graph
```

In [5]:
```
# What is the optimal bipartition?
gplot(G, nodelabel=1:n, edgelabel=wn)
```

Out[5]:



```
In [6]: # Now some functions
        function my_weight_matrix(src::Array,dst::Array,weights::Array)
            n=nv(G)
            sparse([src;dst],[dst;src],[weights;weights],n,n)
        end

        my_laplacian(W::AbstractMatrix)=sparse(diagm(vec(sum(W,2))))-W

        function my_normalized_laplacian(L::AbstractMatrix)
            D=1.0./sqrt.(diag(L))
            n=length(D)
            [L[i,j]*(D[i]*D[j]) for i=1:n, j=1:n]
        end

Out[6]: my_normalized_laplacian (generic function with 1 method)

In [7]: W=my_weight_matrix(sn,tn,wn)

Out[7]: 7×7 SparseMatrixCSC{Int64,Int64} with 22 stored entries:
          [2, 1]  =  2
          [3, 1]  =  3
          [4, 1]  =  4
          [1, 2]  =  2
          [4, 2]  =  7
          [5, 2]  =  1
          [1, 3]  =  3
          [4, 3]  =  3
          [6, 3]  =  2
          [7, 3]  =  1
          ⋮
          [2, 4]  =  7
          [3, 4]  =  3
          [2, 5]  =  1
          [6, 5]  =  7
          [7, 5]  =  3
          [3, 6]  =  2
          [5, 6]  =  7
          [7, 6]  =  5
          [3, 7]  =  1
          [5, 7]  =  3
          [6, 7]  =  5

In [8]: full(W)

Out[8]: 7×7 Array{Int64,2}:
         0  2  3  4  0  0  0
         2  0  0  7  1  0  0
```

```
 3  0  0  3  0  2  1
 4  7  3  0  0  0  0
 0  1  0  0  0  7  3
 0  0  2  0  7  0  5
 0  0  1  0  3  5  0
```

In [9]:
```
L=my_laplacian(W)
full(L)
```

Out[9]:
```
7×7 Array{Int64,2}:
  9  -2  -3  -4   0   0   0
 -2  10   0  -7  -1   0   0
 -3   0   9  -3   0  -2  -1
 -4  -7  -3  14   0   0   0
  0  -1   0   0  11  -7  -3
  0   0  -2   0  -7  14  -5
  0   0  -1   0  -3  -5   9
```

In [10]:
```
Ln=my_normalized_laplacian(L)
```

Out[10]:
```
7×7 Array{Float64,2}:
  1.0       -0.210819   -0.333333  ...   0.0         0.0         0.0
 -0.210819   1.0         0.0           -0.0953463   0.0         0.0
 -0.333333   0.0         1.0            0.0        -0.178174  -0.111111
 -0.356348  -0.591608   -0.267261       0.0         0.0         0.0
  0.0       -0.0953463   0.0            1.0        -0.564076  -0.301511
  0.0        0.0        -0.178174  ...  -0.564076    1.0        -0.445435
  0.0        0.0        -0.111111      -0.301511   -0.445435   1.0
```

In [11]:
```
issymmetric(Ln)
```

Out[11]:
```
true
```

In [12]:
```
# Let us compute the incidence matrix
function my_incidence_matrix(G::Graph, weights::Array)
    A=zeros(nv(G),ne(G))
    k=1
    for a in edges(G)
        A[a.dst,k]=sqrt.(weights[k])
        A[a.src,k]=-sqrt(weights[k])
        k+=1
    end
    A
end
```

Out[12]:
```
my_incidence_matrix (generic function with 1 method)
```

```
In [13]: Iγ=my_incidence_matrix(G,wn)
```

```
Out[13]: 7×11 Array{Float64,2}:
          -1.41421  -1.73205  -2.0   0.0       ...   0.0   0.0       0.0       0.0
           1.41421   0.0       0.0  -2.64575       0.0   0.0       0.0       0.0
           0.0       1.73205   0.0   0.0          -1.0   0.0       0.0       0.0
           0.0       0.0       2.0   2.64575       0.0   0.0       0.0       0.0
           0.0       0.0       0.0   0.0           0.0  -2.64575  -1.73205   0.0
           0.0       0.0       0.0   0.0       ...   0.0   2.64575   0.0      -2.23607
           0.0       0.0       0.0   0.0           1.0   0.0       1.73205   2.23607
```

1.3.3 Facts

1. $L = I_G I_G^T$.
2. L is symmetric PSD matrix.
3. $L\mathbf{1} = 0$ for $\mathbf{1} = [1, ..., 1]^T$, thus 0 is an eigenvalue of L and $\mathbf{1}$ is the corresponding eigenvector.
4. If G has c connected components, then L has c eigenvalues equal to 0.
5. For every $x \in \mathbb{R}^n$, it holds $x^T L x = \sum_{i<j} W_{ij}(x_i - x_j)^2$.
6. For every $x \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$, it holds $(\alpha x + \beta\mathbf{1})^T L(\alpha x + \beta\mathbf{1}) = \alpha^2 x^T L x$.
7. Assume that the eigenvalues of L are increasingly ordered. Then,
$$0 = \lambda_1(L) \leq \lambda_2(L) \leq \cdots \leq \lambda_n(L) \leq 2 \max_{i=1,\cdots,n} d_i.$$
8. $\sigma(L_n) \subseteq [0, 2]$.

1.3.4 Examples

```
In [14]: # Fact 1
         vecnorm(L-Iγ*Iγ')
```

```
Out[14]: 3.607797795906347e-15
```

```
In [15]: # Facts 2 and 7
         issymmetric(L), eigs(L)[1], 2*maximum(diag(L))
```

```
Out[15]: (true, [20.3183, 20.0518, 12.6913, 12.4888, 8.50014, 1.9497], 28)
```

```
In [16]: # Fact 3
         L*ones(n)
```

```
Out[16]: 7-element Array{Float64,1}:
          0.0
          0.0
          0.0
          0.0
          0.0
          0.0
          0.0

In [17]: # Fact 5
         x=rand(n)
         x'*L*x, sum([W[i,j]*(x[i]-x[j])^2 for i=1:n, j=1:n])/2

Out[17]: (8.820471897028595, 8.820471897028597)

In [18]: # Fact 6
         α,β=rand(),rand()
         (α*x+β*ones(n))'*L*(α*x+β*ones(n)), α^2*x'*L*x

Out[18]: (6.990540525195519, 6.990540525195518)

In [19]: # Fact 8
         eigvals(Ln)

Out[19]: 7-element Array{Float64,1}:
          2.18575e-16
          0.17559
          0.887695
          1.29
          1.3692
          1.62123
          1.65628
```

1.4 Bipartitioning

1.4.1 Definitions

Let $\pi = \{V_1, V_2\}$ be a partition of V with $V_1, V_2 \neq \emptyset$.

A **cut** of a partition π is the sum of weights of all edges between V_1 and V_2,

$$cut(\pi) \equiv cut(V_1, V_2) = \sum_{\substack{i \in V_1 \\ j \in V_2}} W_{ij}.$$

A **weight** of a vertex $i \in V$ is the sum of the weights of all egdges emanating from i, $\omega(i) = \sum_{j=1}^{n} W_{ij}$.

A **weight** of a subset $\bar{V} \subset V$ is the sum of the weights of all vertices in $\bar{V}$, $\omega(\bar{V}) = \sum_{i \in \bar{V}} \omega(i)$.

A **proportional cut** of a partition π is

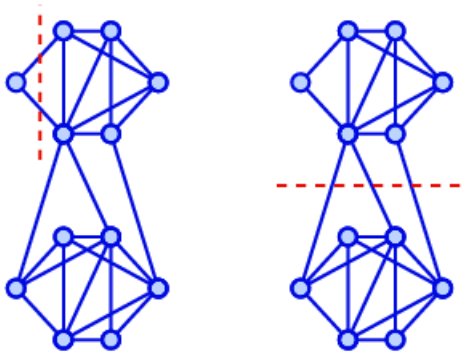
$$pcut(\pi) = \frac{cut(\pi)}{|V_1|} + \frac{cut(\pi)}{|V_2|}.$$

A **normalized cut** of a partition π is

$$ncut(\pi) = \frac{cut(\pi)}{\omega(V_1)} + \frac{cut(\pi)}{\omega(V_2)}.$$

1.4.2 Example

Consider the following partitions (all edges have unit weights):



Two partitions

The left partition π v.s. the right partition π':
$cut(\pi) = 2$ v.s. $cut(\pi') = 3$
$pcut(\pi) = \frac{2}{1} + \frac{2}{11} = 2.18$ v.s. $pcut(\pi') = \frac{3}{6} + \frac{3}{6} = 1$
$ncut(\pi) = \frac{2}{2} + \frac{2}{50} = 1.04$ v.s. $ncut(\pi') = \frac{3}{27} + \frac{3}{25} = 0.23$

1.4.3 Facts

1. The informal description of the bipartitioning problem can be formulated as two problems,

   $$\arg\min_{\pi} pcut(\pi) \quad \text{or} \quad \arg\min_{\pi} ncut(\pi).$$

   The first problem favors partitions into subsets with similar numbers of vertices, while the second problem favors partitions into subsets with similar weights.

2. Both problems are NP-hard.

3. Approximate solutions can be computed by suitable relaxations in $O(n^2)$ operations.

4. The partition π is defined by the vector y such that

$$y_i = \begin{cases} \frac{1}{2} & \text{for } i \in V_1 \\ -\frac{1}{2} & \text{for } i \in V_2 \end{cases}$$

The proportional cut problem can be formulated as the **discrete** proportional cut problem

$$\min_{\substack{y_i \in \{-1/2, 1/2\} \\ \left|\mathbf{y}^T \mathbf{1}\right| \leq \beta}} \frac{1}{2} \sum_{i,j} (y_i - y_j)^2 W_{ij}.$$

Parameter β controls the number of vertices in each subset.

5. The normalized cut problem can be formulated as the **discrete** normalized cut problem

$$\min_{\substack{y_i \in \{-1/2, 1/2\} \\ \left|y^T D \mathbf{1}\right| \leq \beta}} \frac{1}{2} \sum_{i,j} (y_i - y_j)^2 W_{ij}.$$

Parameter β controls the weights of each subset.

6. Using the Fact 5 above, the discrete proportional cut problem can be formulated as the **relaxed** proportional cut problem

$$\begin{gathered} \min_{y \in \mathbb{R}^n} \quad y^T L y. \\ \left|y^T \mathbf{1}\right| \leq 2 \frac{\beta}{\sqrt{n}} \\ y^T y = 1 \end{gathered}$$

Similarly, the discrete normalized cut problem can be formulated as the **relaxed** normalized cut problem

$$\begin{gathered} \min_{y \in \mathbb{R}^n} \quad y^T L_n y. \\ \left|y^T D \mathbf{1}\right| \leq \frac{\beta}{\sqrt{\theta n}} \\ y^T D y = 1 \end{gathered}$$

7. **The Main Theorem.** Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix with eigenvalues $\lambda_1 < \lambda_2 < \lambda_3 \leq \cdots \leq \lambda_n$ and let $v^{[1]}, v^{[2]}, \ldots, v^{[n]}$ be the corresponding eigenvectors. For the fixed $0 \leq \alpha < 1$, the solution of the problem

$$\begin{gathered} \min_{y \in \mathbb{R}^n} \quad y^T A y \\ \left|y^T v^{[1]}\right| \leq \alpha \\ y^T y = 1 \end{gathered}$$

is $y = \pm \alpha v^{[1]} \pm \sqrt{1 - \alpha^2} v^{[2]}$. *(For the proof see [HK04].)*

8. For $0 \leq \beta < \frac{n}{2}$, the solution of the relaxed proportional cut problem is

$$y = \pm \frac{2\beta}{n\sqrt{n}} \mathbf{1} \pm \sqrt{1 - 4\frac{\beta^2}{n^2}} v^{[2]},$$

where $v^{[2]}$ is an eigenvector corresponding to $\lambda_2(L)$. $v^{[2]}$ the **Fiedler vector**. Since the first summand carries no information, V is partitioned according to the signs of the components of $v^{[2]}$:

$$V_1 = \{i : v_i^{[2]} < 0\}, \quad V_2 = \{i : v_i^{[2]} \geq 0\}.$$

Notice that the value of β is irrelevant for the solution.

9. For $0 \leq \beta < \sqrt{\theta n} \left\| D^{\frac{1}{2}}\mathbf{1} \right\|_2$, the solution of the relaxed proportional cut problem is

$$y = \pm \frac{\beta}{\sqrt{\theta n} \left\| D^{\frac{1}{2}}\mathbf{1} \right\|_2^2} \mathbf{1} \pm \sqrt{1 - \frac{\beta^2}{\theta n \left\| D^{\frac{1}{2}}\mathbf{1} \right\|_2^2}} D^{-\frac{1}{2}} v^{[2]},$$

where $v^{[2]}$ is an eigenvector corresponding to $\lambda_2(L_n)$. V is partitioned according to the signs of the components of $v^{[2]}$, as above.

10. Neither of the relaxed algorithms is guaranteed to solve exactly the true (proportional / normalized) cut problem. However, the computed solutions are in the right direction. Whether to use proportional or normalized cut formulation, depends upon the specific problem.

```
In [20]: # Voila!
         λ,v=eigs(L,nev=2,which=:SM, v0=ones(n))

Out[20]: ([2.53765e-16, 1.9497], [-0.377964 -0.383259; -0.377964 -0.373084; ... ; -0.377964 0.40

In [21]: v

Out[21]: 7×2 Array{Float64,2}:
          -0.377964  -0.383259
          -0.377964  -0.373084
          -0.377964  -0.145189
          -0.377964  -0.380089
          -0.377964   0.423708
          -0.377964   0.408506
          -0.377964   0.449408

In [22]: λ,v=eigs(Ln,nev=2,which=:SM, v0=ones(n))

Out[22]: ([2.62948e-17, 0.175846], [-0.344124 0.338025; -0.362738 0.361066; ... ; -0.429198 -0.4

In [23]: v

Out[23]: 7×2 Array{Float64,2}:
          -0.344124   0.338025
          -0.362738   0.361066
          -0.344124   0.126062
          -0.429198   0.45732
          -0.380443  -0.413108
          -0.429198  -0.457099
          -0.344124  -0.388252
```

1.4.4 Example - Concentric circles

A **complete graph** has edges connecting each pair of vertices.

To a set of points $X = \{x_1, x_2, \cdots, x_m\}$, where $x_i \in \mathbb{R}^n$, we assign a weighted complete graph $G = (V, E)$ with m vertices, where the vertex $j \in V$ corresponds to the point $x_j \in X$.

The main idea is to assign weight of an edge $e = (i, j)$ which reflects the distance between x_i and x_j, something like $\omega(e) = \dfrac{1}{\text{dist}(x_i, x_j)}$.

However, this has to be implemented with care. For example, using simple Euclidean distance yield the same results as the function `kmeans()`. In this example we use Gaussian kernel, that is

$$\omega(e) = e^{-\|x_i - x_j\|_2^2/\sigma^2},$$

where the choice of σ is based on experience.

The computation of various distances is implemented in the package Distances.jl.

We will construct the Laplace matrix directly.

```
In [24]: using Winston
         using Colors
         using Distances
```

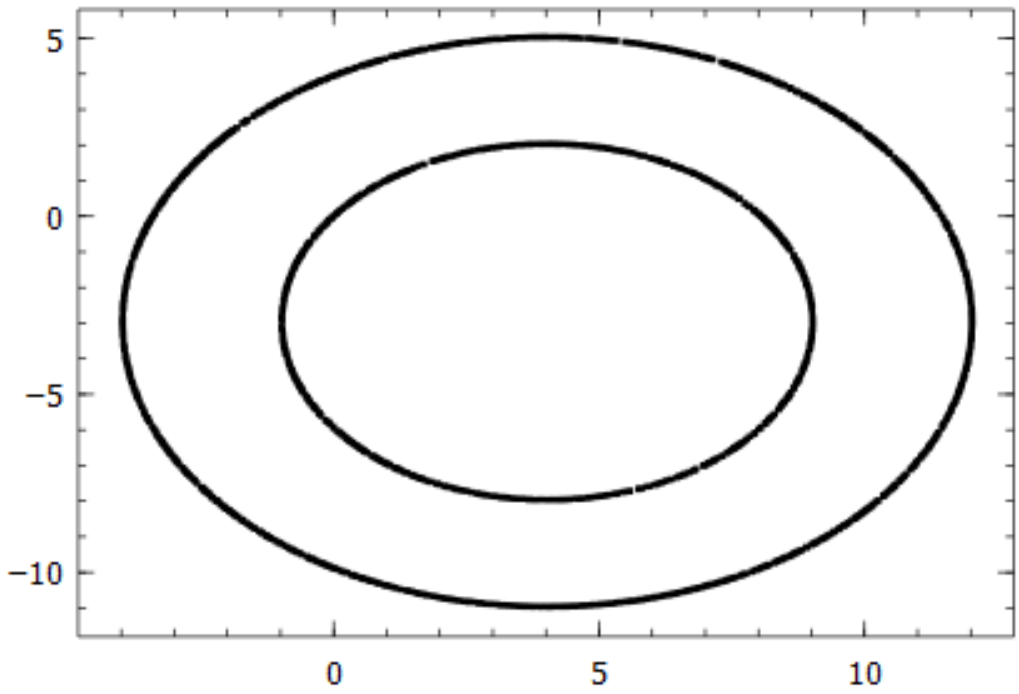
```
In [25]: # Generate two concentric rings
         k=2
         s=srand(421)
         # Center
         center=[rand(-5:5);rand(-5:5)]
         # Radii
         radii=randperm(10)[1:k]
         # Number of points in circles
         sizes=rand(1000:2000,k)
         center,radii,sizes
```

```
Out[25]: ([4, -3], [8, 5], [1932, 1087])
```

```
In [26]: # Points
         m=sum(sizes)
         X=Array{Float64}(2,m)
         first=0
         last=0
         for j=1:k
             first=last+1
             last=last+sizes[j]
             # Random angles
             ϕ=2*π*rand(sizes[j])
             for i=first:last
                 l=i-first+1
```

```
            X[:,i]=center+radii[j]*[cos(ϕ[l]);sin(ϕ[l])]+(rand(2)-0.5)/50
        end
    end
    Winston.plot(X[1,:],X[2,:],".")
```

Out[26]:



In [27]:
```
# Weight matrix
W=1./pairwise(SqEuclidean(),X)
```

Out[27]:
```
3019×3019 Array{Float64,2}:
 Inf             0.00418092    0.0261603    ...    0.00926805    0.00881127
   0.00418092  Inf             0.00398155         0.0292591     0.0333644
   0.0261603     0.00398155  Inf                  0.00633556    0.00623409
   0.0109957     0.0100555     0.00528231         0.0502006     0.0431003
   0.00425747    3.42547       0.00394519         0.0328617     0.0376928
   0.0116045     0.00446525    0.0802536    ...    0.00592752    0.00592787
   0.0150258     0.00421916    0.205658           0.00600517    0.00596938
   0.491721      0.00402425    0.042678           0.00824227    0.00790531
   0.00469215    0.14656       0.0039043          0.0544282     0.0632514
   0.0259226     0.00398746  843.465              0.0063357     0.0062352
   0.00396635    0.234242      0.00419201   ...    0.0195692     0.0217929
```

```
0.0646146     0.00512018    0.0108097        0.01463       0.0134907
0.00503543    0.00836607    0.00986126       0.00695682    0.007199
⋮                                    ⋱
0.00697316    0.0141922     0.0116921        0.0128946     0.0134892
0.00609877    0.100775      0.00617848       0.0457081     0.0532069
0.00777373    0.0112539     0.0148544        0.0115604     0.0119633
0.0111107     0.0206354     0.00681828  ...    0.802728      0.452353
0.00671011    0.0158574     0.0107079        0.0136019     0.0142996
0.0195626     0.00638114    0.0818022        0.0101514     0.0100957
0.00984592    0.00854596    0.0240607        0.0103933     0.0105958
0.00661314    0.0167692     0.0103172        0.0140069     0.014761
0.0783897     0.00599041    0.0544169  ...    0.0131906     0.0127004
0.00735273    0.0125134     0.0131619        0.0121422     0.012629
0.00926805    0.0292591     0.00633556     Inf             7.06324
0.00881127    0.0333644     0.00623409       7.06324     Inf
```

In [28]:
```
# Laplacian matrix
for i=1:m
    W[i,i]=0
end
L=diagm(vec(sum(W,2)))-W
# Check Fact 3
norm(L*ones(m))
```

Out[28]:
```
5.713090892454398e-8
```

In [29]:
```
# Notice λ_1
λ,v=eigs(L,nev=2,which=:SM, v0=ones(m))
```

Out[29]:
```
([-3.01755e-11, 29.5083], [0.0181999 -0.0190234; 0.0181999 0.00830624; ... ; 0.0181999
```

In [30]:
```
# Define clusters
C=ones(Int64,m)
C[find(v[:,2].>0)]=2
```

Out[30]:
```
2
```

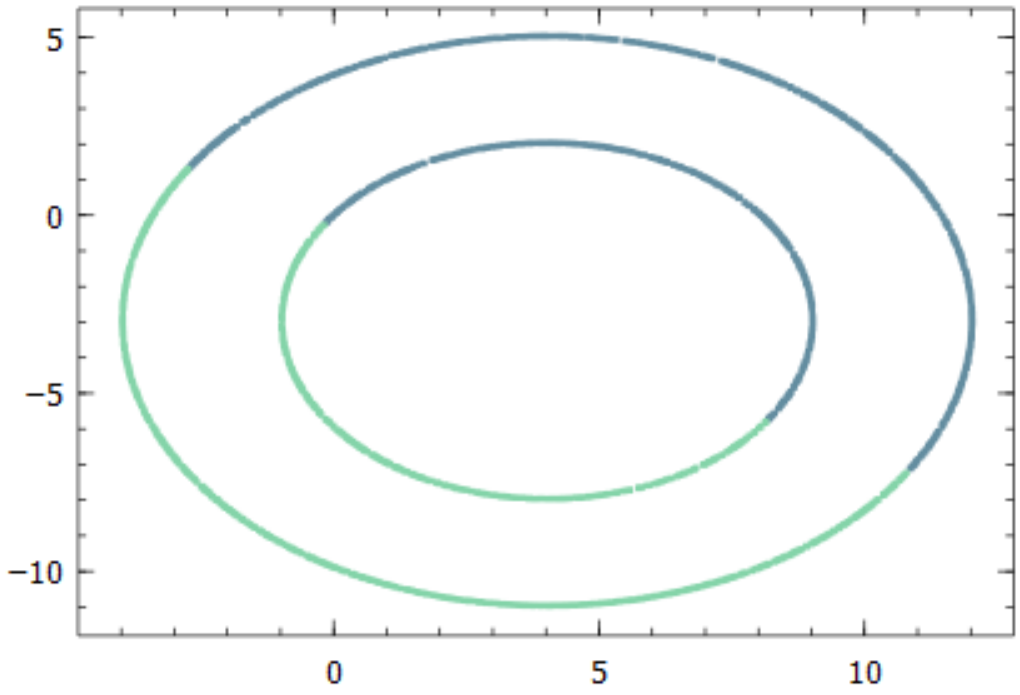
In [31]:
```
# Yet another plotting function
function plotKpartresult(C::Vector,X::Array)
    p=FramedPlot()
    k=maximum(C)
    for j=1:k
        # Random color
        col=RGB(rand(),rand(),rand())
        p1=Points(X[1,find(C.==j)],
        X[2,find(C.==j)],"color",col,symbolkind="dot")
```

```
            add(p,p1)
        end
        p
    end
```

```
Out[31]: plotKpartresult (generic function with 1 method)
```

```
In [32]: plotKpartresult(C,X)
```

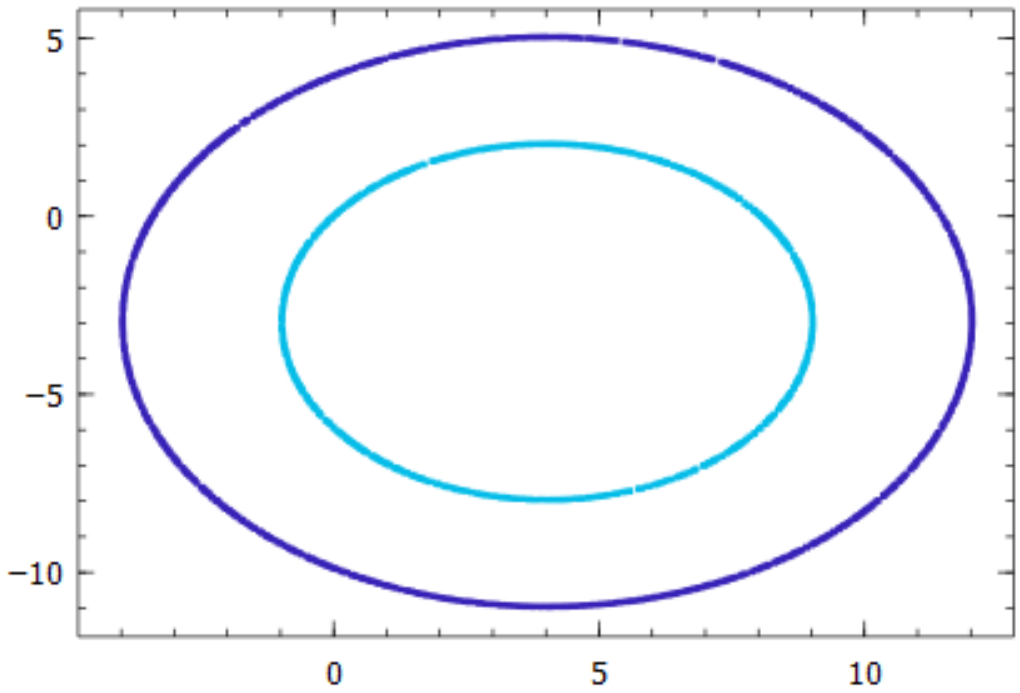
```
Out[32]:
```



This is the same partitioning as obtained by `kmeans()`. Let us try Gaussian kernel. A rule of thumb is: if rings are close, use $\sigma < 1$, if rings are apart, use $\sigma > 1$.

```
In [33]: σ=1.0 # 0.1
         W=exp.(-pairwise(SqEuclidean(),X)/σ^2)-I
         L=diagm(vec(sum(W,2)))-W
         λ,v=eigs(L,nev=2,which=:SM, v0=ones(m))
         C=ones(Int64,m)
         C[find(v[:,2].>0)]=2
         plotKpartresult(C,X)
```

```
Out[33]:
```



1.5 Recursive bipartitioning

1.5.1 Definitions

Let $G = (V, E)$ be a weighted graph with weights ω.

Let $\pi_k = \{V_1, V_2, ..., V_k\}$ be a k-partition of V, with $V_i \neq \emptyset$ for $i = 1, \ldots, k$.

The previous definition of $cut(\pi) \equiv cut(\pi_2)$ extends naturally to k-partition. A **cut** of a partition π_k is

$$cut(\pi_k) = \sum_{i<j} cut(V_i, V_j),$$

where $cut(V_i, V_j)$ is interpreted as a cut of the bipartition of the subgraph of G with vertices $V_1 \cup V_2$.

A **proportional cut** of a partition π_k is

$$pcut(\pi_k) = \sum_{\substack{i,j=1 \\ i<j}}^{k} \left(\frac{cut(V_i, V_j)}{|V_i|} + \frac{cut(V_i, V_j)}{|V_j|} \right) = \sum_{i=1}^{k} \frac{cut(V_i, V \backslash V_i)}{|V_i|}.$$

A **normalized cut** of a partition π_k is

$$ncut(\pi_k) = \sum_{\substack{i,j=1 \\ i<j}}^{k} \left(\frac{cut(V_i, V_j)}{\omega(V_i)} + \frac{cut(V_i, V_j)}{\omega(V_j)} \right) = \sum_{i=1}^{k} \frac{cut(V_i, V \backslash V_i)}{\omega(V_i)}.$$

1.5.2 Fact

If we want to cluster vertices of graph $G = (V, E)$ into k clusters, we can apply the following recursive algorithm:

1. **Initialization.** Compute the bipartition $\pi = \{V_1, V_2\}$ of V. Set the counter $c = 2$.
2. **Recursion.** While $c < k$ repeat:
    1. Compute the bipartition of each subset of V.
    2. Among all $(c+1)$-partitions, choose the one with the smallest $pcut(\pi_{c+1})$ or $ncut(\pi_{c+1})$, respectively.
    3. Set $c = c + 1$.
3. **Stop.**

There is no guarantee for optimality of this algorithm. Clearly, the optimal k-partiton may be a subpartition of one of the discarded partitions.

```
In [ ]:
```