Michael Leibert
Math 504
Homework 9

2. This problem provides an example of how interpolation can be used. The attached spreadsheet provides life expectancy data for the US population. The second column gives the probability of death for the given age. So, for example, the probability that a person between the ages of 20 and 21 dies is 0.000894.

   Suppose a 40 year old decides to buy life insurance. The 40 year old will make monthly payments of $200 every month until death. In this problem we will consider the worth of these payments, a quantity of interest to the insurance company. The payoff upon death will not be considered in this problem. If we assume (continuous time) interest rates of 5% and let m be the number of months past age 40 that the person lives, then the present value of the payments (how much future payments are worth in today's dollars) is,

$$PV = \sum_{i=1}^{m} 200e^{-.05i/12} \tag{1}$$

   Our goal is to determine the average of PV, in other words $E[\text{PV}]$. For the insurance company, this is one way to measure the revenue brought in by the policy. The difficulty is that our data is yearly, while payments are made monthly and people do not always die at the beginning of the month.

   (a) Let $L(t)$ be the probability the 40 year old lives past the age $40 + t$ where t is any positive real number.

```
life<-read.csv("life.csv")
head(life)
```

```
##   Age     P.die
## 1 0-1 0.0068651
## 2 1-2 0.0004689
## 3 2-3 0.0003370
## 4 3-4 0.0002540
## 5 4-5 0.0001937
## 6 5-6 0.0001775
```

```
life$survive<-0;life$die<-0
life$survive[1]<-100000
life$die[1]<-round( life$survive[1]*life[1,2] ) #£

#Start with a population of 100,000
for(i in 2:nrow(life) ){
        life$survive[i]<- life$survive[i-1] - life$die[i-1]
        life$die[i]<- round( life$survive[i]*life[i,2] )        }
head(life)
```

```
##   Age     P.die survive die
## 1 0-1 0.0068651  100000 687
## 2 1-2 0.0004689   99313  47
## 3 2-3 0.0003370   99266  33
## 4 3-4 0.0002540   99233  25
## 5 4-5 0.0001937   99208  19
## 6 5-6 0.0001775   99189  18
```

```
#get the conditional data set, people older than 40

forty<-life[41:nrow(life),]
rownames(forty)<-NULL
forty$Lt<-forty$survive / forty[1,3]
forty$ages<-40:(nrow(forty)+39)     #£
```

```
dat<-data.frame(40:70,forty$Lt[1:31] , " ", " ",71:101, c(forty$Lt[32:nrow(forty)],0) )
names(dat)<-c("ages 40-70", "L(t)" , "...", "...", "ages 71-100", "L(t)" )
dat[nrow(dat),ncol(dat)]<-NA;
```

Estimate $L(t)$ by first considering $t = 0, 1, 2, \ldots$. These values of $L(t)$ can be computed using the spreadsheet data. (For example, for the 40 year old to live to 42, they must not die between the ages $40 - 41$ and $41 - 42$).

```
dat #Estimate  L(t)  by first considering  t=0,1,2,...
```

	ages 40-70	L(t)	...	...	ages 41-100	L(t)
1	40	1.0000000			71	0.76636803
2	41	0.9979572			72	0.74728841
3	42	0.9957589			73	0.72680893
4	43	0.9933844			74	0.70479479
5	44	0.9908128			75	0.68116303
6	45	0.9880234			76	0.65586180
7	46	0.9849955			77	0.62895332
8	47	0.9817188			78	0.60046869
9	48	0.9781932			79	0.57045978
10	49	0.9744188			80	0.53901989
11	50	0.9703747			81	0.50627346
12	51	0.9660507			82	0.47239677
13	52	0.9614156			83	0.43759721
14	53	0.9564487			84	0.40213401
15	54	0.9511396			85	0.36631826
16	55	0.9454572			86	0.33048176
17	56	0.9393807			87	0.29501856
18	57	0.9328792			88	0.26031233
19	58	0.9258902			89	0.22678820
20	59	0.9183309			90	0.19485058
21	60	0.9101288			91	0.16488314
22	61	0.9011904			92	0.13722806
23	62	0.8914328			93	0.11216533
24	63	0.8808457			94	0.08989195
25	64	0.8694913			95	0.07051162
26	65	0.8574006			96	0.05404509
27	66	0.8445322			97	0.04039901
28	67	0.8309069			98	0.02938676
29	68	0.8163794			99	0.02075945
30	69	0.8008358			100	0.01421639
31	70	0.7841826			101	NA

For other t values, interpolate using a cubic spline. In R you can use the **spline** and **splinefun** commands to construct cubic splines, see the help documentation.

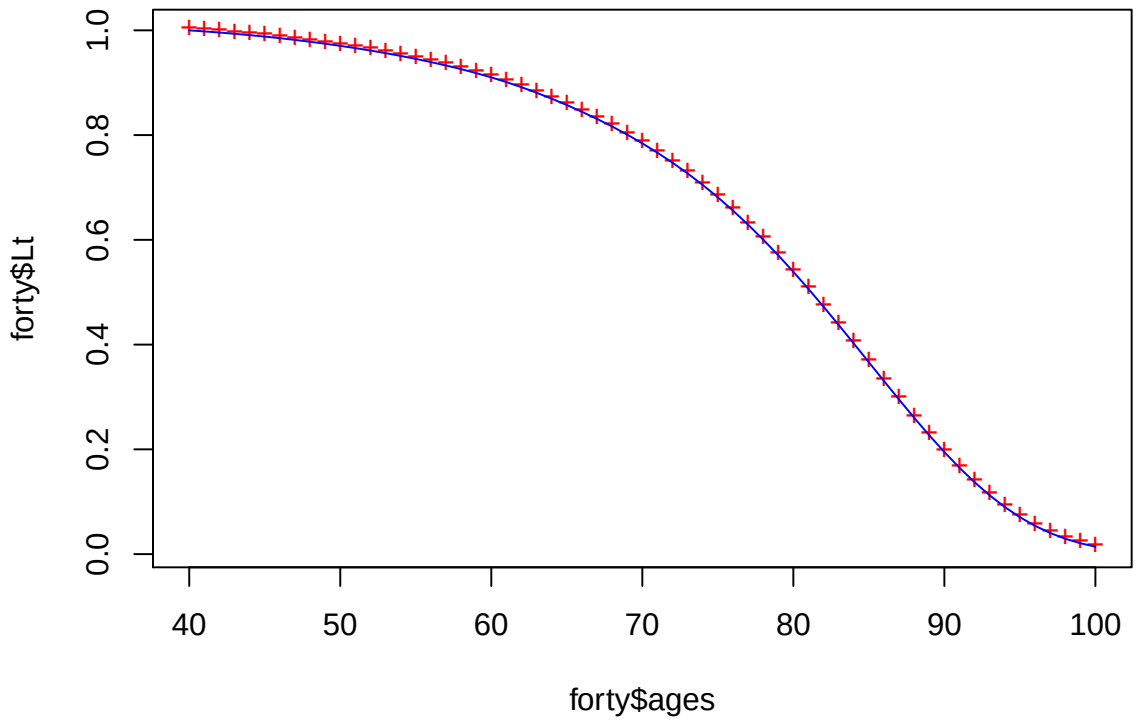
```
func = splinefun(x=forty$ages, y=forty$Lt, method="fmm",  ties = mean)

# 721 values if done by month from age 40 to 100.
# Will show the head and tail of values.
# We can see that the interpolation values at ages 40, 41, 99, and 100 match up.
head( func(seq(40,100,1/12)) ,13)
##  [1] 1.0000000 0.9998352 0.9996695 0.9995028 0.9993351 0.9991664 0.9989968
##  [8] 0.9988261 0.9986544 0.9984817 0.9983079 0.9981331 0.9979572
tail( func(seq(40,100,1/12)) ,13)
```

```
##  [1] 0.02075945 0.02013882 0.01953248 0.01894025 0.01836195 0.01779741
##  [7] 0.01724647 0.01670893 0.01618464 0.01567341 0.01517508 0.01468946
## [13] 0.01421639
```

Graph the interpolating cubic spline of $L(t)$ and include the datapoints, i.e. $L(t)$ for $t = 0, 1, \ldots$..

```
par(mar=c(4.1,4.1,1,1))
smoothingSpline = smooth.spline(forty$ages, forty$Lt , spar=(1/12))
plot(forty$ages, forty$Lt, pch='+', col="red" )
lines(smoothingSpline, col="blue")
```



(b) Explain why the expected (average) present value of the payments is given by

$$E[PV] = \sum_{i=1}^{\infty} 200 L(i/12) e^{-.05i/12} \tag{2}$$

200 is the monthly payment, $L(i/12)$ is the interpolated probability the 40 year old lives past age 40 for each month i. The $\exp(-.05i/12)$ is the continuous compounding of interest for month i with interest rate 5%. If the 40 year old lives to a very old age, the $E[PV]$ is how much the insurance company will make before payout. However, because money has interest-earning potential we calculate the present value, this accounts for time value of money.

In practice we can't sum to ∞, choose an appropriate cutoff and calculate E[PV].

```
rm(dat);dat<-data.frame(1:(61*12), sort( rep(40:100,12) ) ,rep(1:12,61))
dat<-dat[-(722:nrow(dat)),];names(dat)<-c("month","year","ym")
dat$p.life<-NA;dat[which(dat$ym == 1),ncol(dat)]<-forty$Lt
dat$sp.life<- func(seq(40,100,1/12)) ;dat$EPVi<-NA

# calculate E[PV]
```

```
for( i in 1:nrow(dat) ) { dat$EPVi[i]<- 200 * dat$sp.life[ i] * exp( -.05 * i / 12) }
dat$EPV<-cumsum(dat$EPVi)   #£
tail(dat$EPV)           #£
## [1] 39407.12 39407.28 39407.44 39407.59 39407.73 39407.88
```

E[PV] is about \$39,407.

3. Below, let A be an $n \times k$ matrix. Define the span of A, written span(A), as the span of the column vectors of A. In class we discussed Gram-Schmidt (GS) orthogonalization. Here I just want you to go through and finish the arguments I made in class.

   (a) Given a matrix A, write down the GS iteration that will produce an orthogonal matrix Q with span(Q) = span(A).

   Let A be an $n \times k$ matrix with $Span\,(A) = \{a^{(1)}, a^{(2)}, ..., a^{(k)}\}$. We want to produce an orthonormal matrix Q with $Span\,(A) = Span\,(Q)$. From $Span\,(A)$, let $a^{(1)}, a^{(2)}, ..., a^{(n)}$ be linearly independent vectors where $n \leq k$.

   Consider a one dimensional subspace α_1, where $\alpha_1 = Span\left(a^{(1)}\right)$, and note $a^{(1)}$ is orthogonal to everything because it is the only thing in the set. Now we need to normalize this vector to length 1, call this $\mathbf{q_1}$, where $\mathbf{q_1} = \dfrac{a^{(1)}}{||a^{(1)}||}$. Thus $\{\mathbf{q_1}\}$ is an orthonormal basis for α_1.

   Consider a two dimensional subspace α_2, where $\alpha_2 = Span\left(a^{(1)}, a^{(2)}\right)$. We note $a^{(1)}$ is a linear combination of $\mathbf{q_1}$, so $\alpha_2 = Span\left(\mathbf{q_1}, a^{(2)}\right)$. We have to ensure $Span\left(\mathbf{q_1}, a^{(2)}\right)$ is an orthonormal set.

   Recall $\mathbf{q_1}$ and $a^{(2)}$ are linearly independent, so we want to replace $a^{(2)}$ with another vector that is orthogonal to $\mathbf{q_1}$. The vector we want is $\mathbf{y_2} = a^{(2)} - Proj_{\alpha_1}\left(a^{(2)}\right)$ because we will still be able to generate $a^{(2)}$, with a multiple of $\mathbf{q_1}$ plus a multiple of y_2.

$$a^{(2)} - Proj_{\alpha_1}\left(a^{(2)}\right) = a^{(2)} - c\,\mathbf{q_1} = a^{(2)} - \frac{a^{(2)} \cdot \mathbf{q_1}}{\mathbf{q_1} \cdot \mathbf{q_1}}\,\mathbf{q_1} = a^{(2)} - \left(a^{(2)} \cdot \mathbf{q_1}\right)\mathbf{q_1} = \mathbf{y_2}$$

   Now $\alpha_2 = Span\left(a^{(1)}, a^{(2)}\right) = Span\left(\mathbf{q_1}, a^{(2)}\right) = Span\,(\mathbf{q_1}, \mathbf{y_2})$. Because we can generate $a^{(2)}$ with a linear combination of $\mathbf{q_1}$ and $\mathbf{y_2}$ and now $\mathbf{q_1}$ and $\mathbf{y_2}$ are orthogonal.

   To finish we need to normalize: $\mathbf{q_2} = \dfrac{\mathbf{y_2}}{||\mathbf{y_2}||}$, thus $\alpha_2 = Span\,(\mathbf{q_1}, \mathbf{q_2})$

   Consider a three dimensional subspace α_3, where $\alpha_3 = Span\left(a^{(1)}, a^{(2)}, a^{(3)}\right)$. We note $a^{(1)}$ is a linear combination of $\mathbf{q_1}$, $a^{(2)}$ is a linear combination of $\mathbf{q_2}$, so $\alpha_3 = Span\left(\mathbf{q_1}, \mathbf{q_2}, a^{(3)}\right)$. We have to ensure $Span\left(\mathbf{q_1}, \mathbf{q_2}, a^{(3)}\right)$ is an orthonormal set.

   Recall $\mathbf{q_1}$, $\mathbf{q_2}$, and $a^{(3)}$ are linearly independent, so we want to replace $a^{(3)}$ with another vector that is orthogonal to both $\mathbf{q_1}$ and $\mathbf{q_2}$. The vector we want is $\mathbf{y_3} = a^{(3)} - Proj_{\alpha_2}\left(a^{(3)}\right)$ because we will

still be able to generate $a^{(3)}$, with a linear combination of $\mathbf{q_1}$, $\mathbf{q_2}$, and $\mathbf{y_3}$. The vectors $\mathbf{q_1}$, $\mathbf{q_2}$, and $\mathbf{y_3}$ are orthogonal as well.

$$\mathbf{y_3} = a^{(3)} - Proj_{\alpha_2}\left(a^{(3)}\right) = a^{(3)} - \left(a^{(3)} \cdot \mathbf{q_1}\right)\ \mathbf{q_1} - \left(a^{(3)} \cdot \mathbf{q_2}\right)\ \mathbf{q_2}$$

Now $\alpha_3 = Span\left(a^{(1)}, a^{(2)}, a^{(3)}\right) = Span\left(\mathbf{q_1}, \mathbf{q_2}, a^{(3)}\right) = Span\left(\mathbf{q_1}, \mathbf{q_2}, \mathbf{y_3}\right)$. Because we can generate $a^{(3)}$ with a linear combination of $\mathbf{q_1}$, $\mathbf{q_2}$, and $\mathbf{y_3}$ and now $\mathbf{q_1}$, $\mathbf{q_2}$, and $\mathbf{y_2}$ are orthogonal.

To finish we need to normalize: $\mathbf{q_3} = \dfrac{\mathbf{y_3}}{||\mathbf{y_3}||}$, thus $\alpha_3 = Span\left(\mathbf{q_1}, \mathbf{q_2}, \mathbf{q_3}\right)$.

$\vdots$

Consider a n-dimensional subspace α_n, where $\alpha_n = Span\left(a^{(1)}, a^{(2)}, ..., a^{(n)}\right)$. We note $a^{(1)}$ through $a^{(n-1)}$ are a linear combination of $\mathbf{q_1}$ through $\mathbf{q_{n-1}}$, so $\alpha_n = Span\left(\mathbf{q_1}, \mathbf{q_2}, ..., \mathbf{q_{n-1}}, a^{(n)}\right)$. We have to ensure α_n is an orthonormal set.

Recall $\mathbf{q_1}$, $\mathbf{q_2}$, ... , $\mathbf{q_{n-1}}$, and $a^{(n)}$ are linearly independent, so we want to replace $a^{(n)}$ with another vector that is orthogonal to all vectors $\mathbf{q_1}$ through $\mathbf{q_{n-1}}$. The vector we want is $\mathbf{y_n} = a^{(n)} - Proj_{\alpha_{n-1}}\left(a^{(n)}\right)$ because we will still be able to generate $a^{(n)}$, with a linear combination of $\mathbf{q_1}$ through $\mathbf{q_{n-1}}$, and $\mathbf{y_n}$. The vectors $\mathbf{q_1}$ through $\mathbf{q_{n-1}}$, and $\mathbf{y_n}$ are orthogonal as well.

$$\mathbf{y_n} = a^{(n)} - Proj_{\alpha_{n-1}}\left(a^{(n)}\right) = a^{(n)} - \left(a^{(n)} \cdot \mathbf{q_1}\right)\ \mathbf{q_1} - \left(a^{(n)} \cdot \mathbf{q_2}\right)\ \mathbf{q_2} - ... - \left(a^{(n)} \cdot \mathbf{q_{n-1}}\right)\ \mathbf{q_{n-1}}$$

Now $\alpha_n = Span\left(a^{(1)}, a^{(2)}, ..., a^{(n)}\right) = Span\left(\mathbf{q_1}, \mathbf{q_2}, ..., a^{(n)}\right) = Span\left(\mathbf{q_1}, \mathbf{q_2}, ..., \mathbf{y_n}\right)$. Because we can generate $a^{(n)}$ with a linear combination of $\mathbf{q_1}$ through $\mathbf{q_{n-1}}$, and $\mathbf{y_n}$ and now $\mathbf{q_1}$ through $\mathbf{q_{n-1}}$, and $\mathbf{y_n}$ are orthogonal.

To finish we need to normalize: $\mathbf{q_n} = \dfrac{\mathbf{y_n}}{||\mathbf{y_n}||}$, thus $\alpha_n = Span\left(\mathbf{q_1}, \mathbf{q_2}, ..., \mathbf{q_n}\right)$.

Form the matrix Q with the linearly independent orthornormal vectors $\mathbf{q_1}, \mathbf{q_2}, ..., \mathbf{q_n}$. Q is an orthogonal matrix. $Span(Q) = Span(\alpha_n)$. Since $Span(\alpha_n)$ was formed with n linearly independent columns of A, $Span(\alpha_n) = Span(A)$. Thus $Span(Q) = Span(A)$.

(b) Prove that the GS iteration you wrote down in (a) produces orthonormal vectors with the correct span (see Sauer if you get stuck).

Orthogonal:

$n = 2$

$$\begin{aligned}\mathbf{q_1} \cdot \mathbf{y_2} &= \mathbf{q_1} \cdot \left(a^{(2)} - \left(a^{(2)} \cdot \mathbf{q_1}\right) \mathbf{q_1}\right) \\ &= \mathbf{q_1} \cdot a^{(2)} - \mathbf{q_1} \cdot \mathbf{q_1} \left(\mathbf{q_1} \cdot a^{(2)}\right) \\ &= \mathbf{q_1} \cdot a^{(2)} - (1) \left(\mathbf{q_1} \cdot a^{(2)}\right) \\ &= 0\end{aligned}$$

$n = 3$

$$\begin{aligned}\mathbf{q_1} \cdot \mathbf{y_3} &= \mathbf{q_1} \cdot \left(a^{(3)} - \left(a^{(3)} \cdot \mathbf{q_1}\right) \mathbf{q_1}\right) \\ &= \mathbf{q_1} \cdot a^{(3)} - \mathbf{q_1} \cdot \mathbf{q_1} \left(\mathbf{q_1} \cdot a^{(3)}\right) \\ &= \mathbf{q_1} \cdot a^{(3)} - (1) \left(\mathbf{q_1} \cdot a^{(3)}\right) \\ &= 0\end{aligned} \qquad \begin{aligned}\mathbf{q_2} \cdot \mathbf{y_3} &= \mathbf{q_2} \cdot \left(a^{(3)} - \left(a^{(3)} \cdot \mathbf{q_2}\right) \mathbf{q_2}\right) \\ &= \mathbf{q_2} \cdot a^{(3)} - \mathbf{q_2} \cdot \mathbf{q_2} \left(\mathbf{q_2} \cdot a^{(3)}\right) \\ &= \mathbf{q_2} \cdot a^{(3)} - (1) \left(\mathbf{q_2} \cdot a^{(3)}\right) \\ &= 0\end{aligned}$$

$n = n$, for any vector $i = 1, ..., n-1$

$$\begin{aligned}\mathbf{q_i} \cdot \mathbf{y_n} &= \mathbf{q_i} \cdot \left(a^{(n)} - \left(a^{(n)} \cdot \mathbf{q_i}\right) \mathbf{q_i}\right) \\ &= \mathbf{q_i} \cdot a^{(n)} - \mathbf{q_i} \cdot \mathbf{q_i} \left(\mathbf{q_i} \cdot a^{(n)}\right) \\ &= \mathbf{q_i} \cdot a^{(n)} - (1) \left(\mathbf{q_i} \cdot a^{(n)}\right) \\ &= 0\end{aligned}$$

The vectors were normalized in (a) and dividing by the norm does not effect the orthogonalization $\left(\frac{0}{||y_n||}\right)$.

For our span in (a), we wrote each y_i, $i = 1, ..., n$ as a linear combination of the original vector $a^{(i)}$,

$$\mathbf{y_i} = a^{(i)} - c_1 \ \mathbf{q_1} - c_2 \ \mathbf{q_2} ...$$

Where the c's were: $\frac{a^{(i)} \cdot \mathbf{q_j}}{\mathbf{q_j} \cdot \mathbf{q_j}}$, $j \neq i$. So the span will be the same as $a^{(i)}$.

(c) Write an R function **GramSchmidt(A)** which returns the matrix Q in (a). Check that your function works and compare to the result of using R's **qr** function for some non-trivial choice of A.

```
a1<-sample(3:6,1);a2<-sample(3:6,1)
A<-matrix(sample(0:9,a1*a2,replace=T),a1,a2)
A
##      [,1] [,2] [,3] [,4]
## [1,]    5    5    7    4
## [2,]    3    7    4    5
## [3,]    8    5    4    7
GramSchmidt<-function(A){                                    #}
        O<-A
        #get rid of linearly dependent columns
        A<-A[, qr(A)$pivot[seq_len(qr(A)$rank)]]
```

```
        #matrix Q
        Q<-matrix(NA,dim(A)[1],dim(A)[2])

        Q[,1] <- as.matrix(A[,1]) / norm(as.matrix(A[,1]) ,type="2")

        for( i in 2:ncol(A) ) {
                a<-as.matrix( A[,i] )
        for( j in  (i-1):1 ) {
                a<- a - (  ( t(a) %*% Q[,j]  ) *  Q[,j]  ) }
        Q[,i]<-a/norm(a,type="2") }

        #R gives a orthognoal matrix, need to normalize columns
        QQ<-qr.Q( qr(O) )
        for( i in 1:ncol(QQ) ) { QQ[,i] <-  QQ[,i] /
                norm(as.matrix(QQ[,i]),type="2") }

        print("Our GS matrix:");print(Q)
        print("R's normalized matrix:");print(QQ)
        all( (abs(round(Q,10)) == abs(round(QQ,10))) == T);
        }
options(warn=-1)
GramSchmidt(A)
## [1] "Our GS matrix:"
##           [,1]       [,2]       [,3]
## [1,] 0.5050763  0.1262143  0.8537962
## [2,] 0.3030458  0.9003287 -0.3123645
## [3,] 0.8081220 -0.4165072 -0.4164859
## [1] "R's normalized matrix:"
##            [,1]       [,2]       [,3]
## [1,] -0.5050763 -0.1262143 -0.8537962
## [2,] -0.3030458 -0.9003287  0.3123645
## [3,] -0.8081220  0.4165072  0.4164859
## [1] TRUE
options(warn=0)
```

4. Attached you will find an article by M Newman published in PNAS in 2006. Read the article. Much of the beginning is introduction, the key quantitative ideas start on p8578, centering on equation [1]-[4]. Explain Newman's idea of detecting community structure. Your explanation should walk through the equations and justify each step. Apply Newman's ideas to the Karate Network (attached) mentioned in the paper. Use power method combined with orthogonalization, as discussed in class, to compute all the eigenvectors of the matrix Newman discusses. (You don't need all the eigenvectors, but compute them all so you can work through the algorithm.) You can use eigen to check if your computations are correct. Plot the network with the community structure you compute shown.

   Newman wants to determine whether there exists any natural division of its vertices into nonoverlapping groups or communities, where these communities may be of any size. He defines good division of a network into communities by one in which there are fewer than expected edges between communities. "If the number of edges between groups is significantly less than we expect by chance, or equivalent if the number within groups is significantly more, then it is reasonable to conclude that something interesting is going on," he says.

   The idea of true community structure in a network corresponding to a statistically surprising arrangement of edges is quantified by modularity. He precisely defines modularity as, "The number of edges falling

within groups minus the expected number in an equivalent network with edges placed at random, up to a multiplicative constant." And the maximization of the modularity is the definitive current method of community detection.

Newman considers a network of n vertices. He divides the network into two groups, where $s_i = 1$ if vertex i is in group 1 and $s_i = -1$ if group i is in group 2. The matrix A is the adjacency matrix, with A_{ij} the number of edges between vertices i and j.

The expected number of edges between vertices i and j if edges are placed at random is $\frac{k_i k_j}{2m}$, where k_i and k_j are the degrees of the vertices, and $m = \frac{1}{2}\sum_i k_i$ is the total number of edges in the network.

The modularity Q is given by the sum of $A_{ij} - \frac{k_i k_j}{2m}$ over all pairs of vertices i, j that fall in the same group. The modularity is expressed by:

$$
\begin{aligned}
Q &= \frac{1}{4m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) (s_i s_j + 1) \\
&= \frac{1}{4m} \sum_{ij} \left[(s_i s_j A_{ij}) - \frac{k_i k_j}{2m} + A_{ij} - \frac{k_i k_j}{2m}\, s_i s_j \right] \\
&\quad \text{The last two terms will cancel out because } 2m = \sum_i k_i = \sum_{ij} A_{ij} \\
&= \frac{1}{4m} \left(\sum_{ij} \left[(s_i s_j A_{ij}) - \frac{k_i k_j}{2m}\, s_i s_j \right] + \cancel{\sum_{ij} A_{ij}} - \cancel{\sum_{ij} \frac{k_i k_j}{2m}} \right) \\
&= \frac{1}{4m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) s_i s_j
\end{aligned}
$$

If we define $A_{ij} - \frac{k_i k_j}{2m}$ as the elements of matrix B and separate the summation into two, we can write:

$$
\begin{aligned}
Q &= \frac{1}{4m} \sum_i \sum_j B_{ij} s_i s_j \qquad \text{this is now in quadratic form.} \\
&= \frac{1}{4m}\, \mathbf{s}^T B \mathbf{s}
\end{aligned}
$$

Now we can write $\mathbf{s} = \left(\mathbf{u}_1^T \cdot \mathbf{s}\right) \mathbf{u}_1 + \left(\mathbf{u}_2^T \cdot \mathbf{s}\right) \mathbf{u}_2 ... + \left(\mathbf{u}_n^T \cdot \mathbf{s}\right) \mathbf{u}_n$ as a linear combination of the normalized eigenvectors u_i of B. Define $\left(\mathbf{u}_i^T \cdot \mathbf{s}\right) = a_i$

$$
\begin{aligned}
Q &= \frac{1}{4m}\,\mathbf{s}^T B\mathbf{s} \\
&= \frac{1}{4m}\left[\left(\mathbf{u}_1^T\cdot\mathbf{s}\right)\mathbf{u}_1+\left(\mathbf{u}_2^T\cdot\mathbf{s}\right)\mathbf{u}_2\ldots+\left(\mathbf{u}_n^T\cdot\mathbf{s}\right)\mathbf{u}_n\right]^T B\left[\left(\mathbf{u}_1^T\cdot\mathbf{s}\right)\mathbf{u}_1+\left(\mathbf{u}_2^T\cdot\mathbf{s}\right)\mathbf{u}_2\ldots+\left(\mathbf{u}_n^T\cdot\mathbf{s}\right)\mathbf{u}_n\right] \\
&= \frac{1}{4m}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(a_i\mathbf{u}_i^T\right)B\left(a_i\mathbf{u}_i\right) \\
&= \frac{1}{4m}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\mathbf{u}_i^T B\mathbf{u}_i\right)a_i^2 \\
&= \frac{1}{4m}\sum_{i=1}^{n}\beta_i a_i^2 \\
&= \frac{1}{4m}\sum_{i=1}^{n}\left(\mathbf{u}_i^T\cdot\mathbf{s}\right)^2\beta_i
\end{aligned}
$$

β_i is the eigenvalue of B corresponding to the eigenvector u_i. Also assume the eigenvectors are in decreasing order, $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$. We want to maximize the modularity by choosing the value $\mathbf{s}$.

Due to the constraint, $\mathbf{s} = \pm 1$, the maximum is achieved by setting $\mathbf{s}_i = 1$ if the corresponding element of $\mathbf{u}_1$ is positive and $\mathbf{s}_i = -1$ otherwise. This placed all vertices whose corresponding elements are positive in one group and all the rest in the other.

```
library(igraph)

A <- read.table("karateclub.txt")
k<- (rowSums(A))
m<-1/2 * sum(rowSums(A))
B<-matrix(NA,dim(A)[1],dim(A)[1])

for( i in 1:nrow(B)){
for( j in 1:ncol(B)){   B[i,j]<-A[i,j] - ( ( sum(A[,i])*sum(A[,j]) )  / (2*m) ) }}

Pi<-function(M){
        x<-matrix(1,ncol(M),ncol(M))
        v<-matrix(NA,ncol(M),ncol(M))
        rq<-rep(NA,ncol(M))

        repeat{
                y<-x
                for( i in 1:ncol(M) ){  v[,i]<-M%*%x[,i] }
                x<-qr.Q(qr(v))
                        if (sum(abs(x[,1]-y[,1])) < 10^-6) { break }}
        for( i in 1:ncol(M) ){rq[i] <-
                t(x[,i]) %*% M %*% x[,i]/sum(x[,i]*x[,i]) }
        mylist<-list(x,round(rq,10))
        return(mylist)
}

Pi(B)[[2]]          #Rayleigh quotients

##  [1] -5.5924963  4.9770802 -3.4511137 -3.1029766  3.0191196 -2.4392002
##  [7]  2.3202047 -2.0906924 -2.0003853 -1.6880573  1.4752798  0.1274896
## [13] -0.1208442 -1.1926023 -0.4262342  0.4313176  1.0313044  0.8381708
```

```
## [19] -0.7929009  0.6167644 -0.4280059  0.4195316  0.3000158  0.0000000
## [25]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
## [31]  0.0000000  0.0000000  0.0000000  0.0000000
```

```
round(eigen(B)$values,5)
```

```
##  [1]  4.97708  3.04278  2.32021  1.48883  1.45942  1.08329  1.03149
##  [8]  0.83817  0.61676  0.41973  0.30002  0.00000  0.00000  0.00000
## [15]  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
## [22]  0.00000 -0.42821 -0.79290 -1.07838 -1.19260 -1.46633 -1.68806
## [29] -2.00000 -2.09108 -2.43920 -3.12664 -3.45111 -5.59250
```

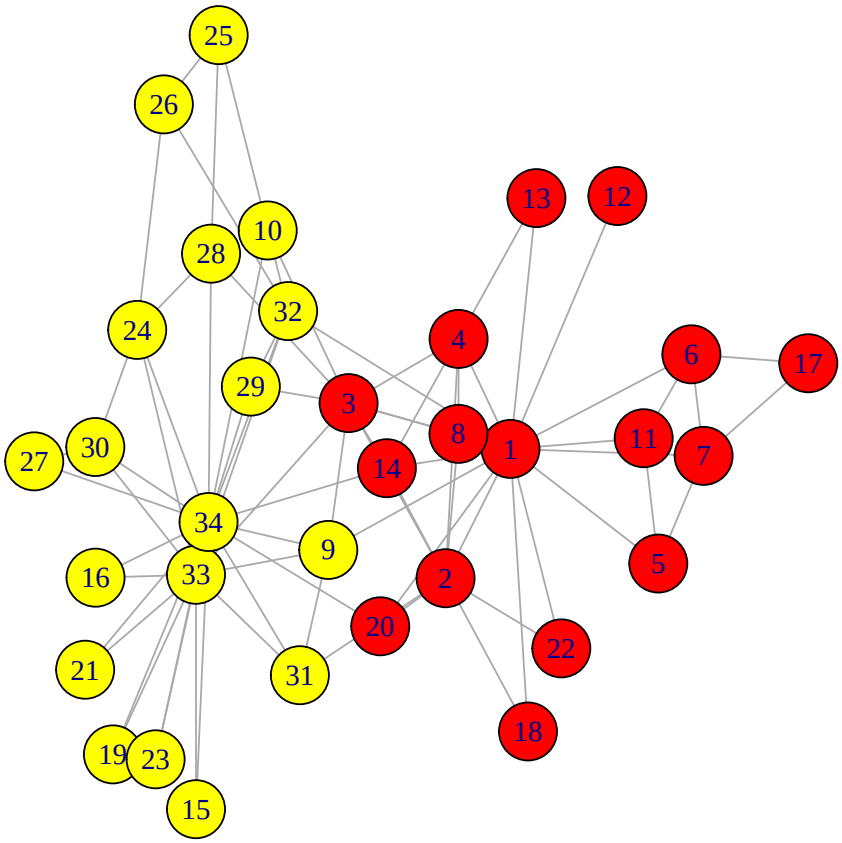
```
Pi(B)[[1]][,2]
```

```
##  [1] -0.38754297 -0.26955744 -0.13189900 -0.25345236 -0.13400323
##  [6] -0.14573804 -0.14573804 -0.20935953  0.05447573  0.04785239
## [11] -0.13400323 -0.07784278 -0.12874396 -0.13502860  0.13943289
## [16]  0.13943289 -0.05851822 -0.13197980  0.13943289 -0.05764887
## [21]  0.13943289 -0.13197980  0.13943289  0.21674713  0.05633049
## [26]  0.07540038  0.11580257  0.10276487  0.06834028  0.20629457
## [31]  0.09626391  0.10185679  0.32390450  0.36983781
```

```
eigen(B)$vector[,1]
```

```
##  [1] -0.38754293 -0.26955742 -0.13189899 -0.25345236 -0.13400323
##  [6] -0.14573804 -0.14573804 -0.20935954  0.05447571  0.04785238
## [11] -0.13400323 -0.07784279 -0.12874396 -0.13502861  0.13943288
## [16]  0.13943288 -0.05851821 -0.13197981  0.13943288 -0.05764888
## [21]  0.13943288 -0.13197981  0.13943288  0.21674712  0.05633050
## [26]  0.07540039  0.11580257  0.10276487  0.06834027  0.20629456
## [31]  0.09626391  0.10185678  0.32390453  0.36983786
```

```
s<-ifelse(Pi(B)[[1]][,2] > 0 , 1,-1)
g=graph.adjacency(as.matrix(A),mode="undirected",weighted=NULL,diag=F)
V(g)$name<-1:(dim(A)[1])
V(g)$color<-ifelse(s == 1, "yellow", "red")
plot.igraph(g)
```


25
26
13
12
10
28
32
24
4
6
17
29
3
8
1
11
7
27
30
14
34
9
5
16
33
2
20
22
21
31
18
19
23
15