Michael Leibert

Math 504

Homework 7

2. In this problem you'll apply power iteration to compute the dominant eigenvector v which has eigenvalue 1 and satisfies $M^T v = v$. (REMEMBER, you need to consider M^T not M!) Remember, the v you compute will have to be normalized to sum to 1, rather than the usual normalization of length 1. If you normalize v in this way then v_i will represent the probability of being at page i after a long time surfing the web. If we sort the pages in descending order of their v_i we get the PageRank ordering.

   (a) To get your code running and for orientation, build a toy network of 5 nodes (pages). Make sure that there is a path from each node in the network to every other node.

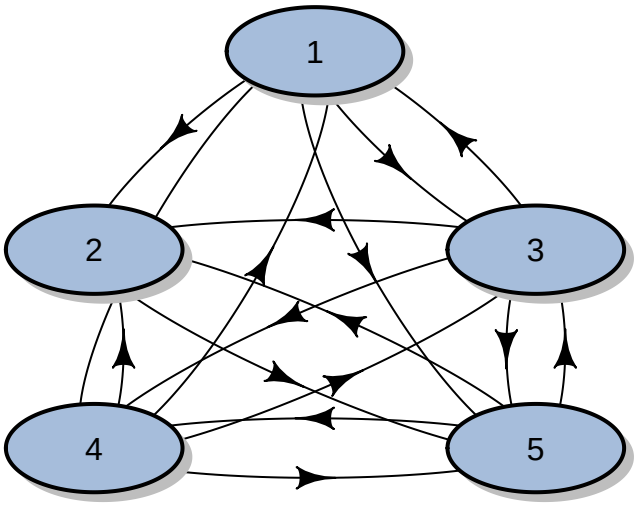
```
mat<-matrix(0,5,5)

for( i in 1:5){
      y<-sample((1:5)[-i],sample(2:4,1))
      x<-sample(1:10,length(y))
      x<-x/sum(x)
      mat[i,y]<-x
}

mat;rowSums(mat)
```

```
##           [,1]      [,2]      [,3]       [,4]      [,5]
## [1,] 0.0000000 0.0000000 0.8181818 0.18181818 0.0000000
## [2,] 0.3333333 0.0000000 0.1111111 0.16666667 0.3888889
## [3,] 0.5000000 0.0000000 0.0000000 0.08333333 0.4166667
## [4,] 0.2000000 0.0000000 0.3000000 0.00000000 0.5000000
## [5,] 0.3214286 0.2857143 0.2142857 0.17857143 0.0000000
```

```
## [1] 1 1 1 1 1
```



(a) Using different values of p, compute the dominant eigenvector v using power iteration and use **eigen** to determine the first two eigenvalues. (Remember to tranpose your matrix!)

```
####### We note we can write B as a vector 1/n * p *  sum x

myfunction<-function(matt,p){
  m=dim(matt)[1]
  x<-sample(1:10,m,replace=T);i=1
  x<-   x/sum(x)
  tt<-system.time(
  repeat{
      y<-x
      i=i+1
      x<- ( (1-p) * t(matt) %*%x ) + ( rep( (1/m)* p * sum(x),m) ) /
          norm(( (1-p) * t(matt) %*%x ) + ( rep( (1/m)* p * sum(x),m) ))
      if( norm(x-y) < 10^-14 ) {break} } )
    print(t(x))
  print(tt)
  print(paste0("Iterations: ", i))

  G<-( (1-p) * matt  + ( p) * matrix(1/m,m,m) )
  print( Re(eigen(t(G))$vectors[,1]) / Re( sum(eigen(t(G))$vectors[,1])  ) )
  print(sort( sqrt(Re(eigen((1-p) *  (matt) + p * 1/m )$values )^2+
   Im(eigen((1-p) *  (matt) + p * 1/m)$values)^2 ) , decreasing = T)[1:2])
  }

myfunction(mat,.05)
```

```
##           [,1]       [,2]      [,3]     [,4]      [,5]
## [1,] 0.2709813 0.06969736 0.3097126 0.129671 0.2199376
##    user  system elapsed
##    0.01    0.00    0.01
## [1] "Iterations: 52"
## [1] 0.27098135 0.06969736 0.30971262 0.12967105 0.21993763
## [1] 1.0000000 0.5680198
```

```
myfunction(mat,.5)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.2361529 0.1312212 0.2516745 0.1624032 0.2185482
##    user  system elapsed
##       0       0       0
## [1] "Iterations: 29"
## [1] 0.2361529 0.1312212 0.2516745 0.1624032 0.2185482
## [1] 1.0000000 0.2989578
```

```
myfunction(mat,.95)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.2035502 0.1928988 0.2045146 0.1961218 0.2029146
##    user  system elapsed
##       0       0       0
## [1] "Iterations: 12"
## [1] 0.2035502 0.1928988 0.2045146 0.1961218 0.2029146
## [1] 1.00000000 0.02989578
```

(a) Discuss how p affects the distance between λ_1 and λ_2 and how this relates to the number of iterations

needed for power iteration. If $\lambda = a + bi$, that is λ is complex, then it's size is $\sqrt{a^2 + b^2}$ and you can compare how far this is from 1.

We know the power method will converge at a rate $\left|\frac{\lambda_2}{\lambda_1}\right|$, so the smaller λ_2 is the faster the convergence and less iterations are required. With a lower p, λ_2 has higher values. For example $p = .05$, $\lambda_2 = 0.5680198$ and we see the amount of iterations required was 52. When we set $p = 0.5$, $\lambda_2 = 0.2989578$ and the convergence was quicker with only 29 iterations. Lastly, when $p = 0.95$ it only took 12 iterations to converge because $\lambda_2 = 0.02989578$ which was considerably smaller than λ_1. For the increasing p the ratio was smaller (smaller numerator), and less iterations were required.

(b) Now consider the Hollins Universtiy network and set $p = .15$. Compute the dominant eigenvector of M^T using power iteration. Try to use **eigen** to compute v; you'll see that eigen doesn't work well here. Determine the highest rated web page.

```
edge<-read.table("edges.txt")
node<-read.table("nodes.txt")

G<-matrix(0,max(edge),max(edge))
n=max(edge)

#pages without links
length( setdiff(edge[,2],edge[,1]) ); head(setdiff(edge[,2],edge[,1]))
```

```
## [1] 3189
```

```
## [1]   3  67  73  91 107 108
```

```
#G as a zero matrix, by row replace 0 entry with 1 if there is an edge
for( i in 1:max(edge) ) {
    if( length( which(edge[,1]  == i)) == 0 ){ G[i,i]<-1;next}
    G[i,edge[ which(edge[,1]  == i) ,2]]<-1 }

#stochastic matrix row must sum to 1
G<- apply(G, 2, "/",  rowSums(G)  )
 all(round(rowSums(G),15) == 1) #yes
```

```
## [1] TRUE
```

```
########


myfunction<-function(matt,p){
  m=dim(matt)[1]
  x<-sample(1:10,m,replace=T);i=1
  x<-  x/sum(x)
  tt<-system.time(
  repeat{
      y<-x
      i=i+1
      x<-( (1-p) * t(matt) %*% x ) + ( rep( (1/m)* p * sum(x),m) ) /
          norm( ( (1-p) * t(matt) %*% x ) + ( rep( (1/m)* p * sum(x),m) ) )
      if( norm(x-y) < 10^-14 ) {break} } )
    x<<-x
  print(tt)
  print(paste0("Iterations: ", i))
```

```
  #remove the following lines, eigen freezes R with a larger matrix
  #G<-( (1-p) * matt  + ( p) * matrix(1/m,m,m) )
  #print( Re(eigen(t(G))$vectors[,1]) / Re( sum(eigen(t(G))$vectors[,1])  ) )
  }

myfunction(G,.15)
```

```
##    user  system elapsed
##  243.90   20.46  266.99
## [1] "Iterations: 182"
```

```
node[,1]<-round(x,6)
head( node[with(node, order(V1,decreasing = T)), ] , 10);sum(node[,1])
```

```
##           V1                                                                V2
## 73  0.009506                                 http://www.hollins.edu/calendar
## 2   0.008543                                         http://www.hollins.edu/
## 593 0.008029                          http://www.hollins.edu/calendar/null.htm
## 37  0.003991                 http://www.hollins.edu/admissions/visit/visit.htm
## 38  0.003700                       http://www.hollins.edu/about/about_tour.htm
## 61  0.003466                           http://www.hollins.edu/htdig/index.html
## 52  0.003449 http://www.hollins.edu/admissions/info-request/info-request.cfm
## 43  0.003079                 http://www.hollins.edu/admissions/apply/apply.htm
## 91  0.002977                   http://www1.hollins.edu/docs/library/libtoc.htm
## 425 0.002829 http://www.hollins.edu/academics/library/resources/web_linx.htm
```

```
## [1] 0.999857
```

```
### Let us use sparsity to speed things up



library(Matrix);rm(x)
edge<-read.table("edges.txt")
nodez<-read.table("nodes.txt")

#Adding Edges
zeroes<-setdiff(edge$V1,edge$V2)
dat<-data.frame(setdiff(edge$V2,edge$V1), setdiff(edge$V2,edge$V1))
names(dat)<-names(edge)
edge<-rbind(edge,dat)
dat<-data.frame(setdiff(edge$V1,edge$V2), setdiff(edge$V1,edge$V2))
names(dat)<-names(edge)
edge<-rbind(edge,dat);rm(dat)


setdiff(edge$V2,edge$V1)
```

```
## integer(0)
```

```
setdiff(edge$V1,edge$V2)
```

```
## integer(0)
```

```
edge$V3<-1
edge[zeroes,3]<-0
tail(edge)
```

```
##         V1   V2 V3
## 27061 6009 6009  1
```

```
## 27062 6010 6010  1
## 27063 6011 6011  1
## 27064 6012 6012  1
## 27065    1    1  1
## 27066   51   51  1
```

```
A <- sparseMatrix(i = edge$V1, j = edge$V2, x = edge$V3)
A <- A/rowSums(A)
 all(round(rowSums(A),14) == 1)
```

```
## [1] TRUE
```

```
myfunction(A,.15)
```

```
##    user  system elapsed
##    0.53    0.00    0.53
## [1] "Iterations: 181"
```

```
nodez[,1]<-round( as.numeric( x ) , 6 ) #much faster
head( nodez[with(nodez, order(V1,decreasing = T)), ] , 10);sum(nodez[,1])
```

```
##           V1                                                                V2
## 73  0.009509                                 http://www.hollins.edu/calendar
## 2   0.008517                                         http://www.hollins.edu/
## 593 0.008031                          http://www.hollins.edu/calendar/null.htm
## 37  0.003993                 http://www.hollins.edu/admissions/visit/visit.htm
## 38  0.003702                       http://www.hollins.edu/about/about_tour.htm
## 61  0.003468                           http://www.hollins.edu/htdig/index.html
## 52  0.003452 http://www.hollins.edu/admissions/info-request/info-request.cfm
## 43  0.003081                 http://www.hollins.edu/admissions/apply/apply.htm
## 91  0.002977                   http://www1.hollins.edu/docs/library/libtoc.htm
## 425 0.002829 http://www.hollins.edu/academics/library/resources/web_linx.htm
```

```
## [1] 0.999854
```

(3) The network has about 300,000 nodes and about 1,000,000 edges. Look at the Notre Dame file to see the data format. Names have been stripped and the nodes in the network are simply numbered. Each row in the datafile contains two numbers, corresponding to a start node and end node connected by a directed edge. You can load the data into R using **read.table** as usual.

As we discussed in class, doing a single iteration of power method is $O(n^2)$. For the Hollins University dataset this is not too bad, but for the Notre Dame network this would be very slow and, besides, we can't store the full A matrix.. However, the adjacency matrix (i.e. A) of most networks will be sparse, meaning most entries will be zero. Both in terms of storage and matrix multiplication, great savings can be achieved by not considering 0 entries. This comes down to representing the matrix A in a data structure different than the usual $n \times n$ grid and exploiting this data structure to reduce the $O(n^2)$ to $O(n)$. We will not consider the specific data structures used to store sparse matrices, but instead simply use sparse matrices through R packages. The **Matrix** package can handle sparse matrices. A nice, short writeup about R packages dealing with sparse matrices is given here

To get a sparse matrix, use the `as_adjacency_matrix` function from igraph with the **sparse** flag set to TRUE.

(a) Try to create an A matrix with and without sparsity for this graph. You will see that without sparsity, R will run out of memory.

```
edge<-read.table("web-NotreDame.txt")

#A<-matrix(0,max(edge),max(edge))
# Error: cannot allocate vector of size 790.5 Gb Execution halted
```

```
#This would be a 0 matrix with the appropriate dimensions

#Adding Edges
dat<-data.frame(setdiff(edge$V2,edge$V1), setdiff(edge$V2,edge$V1))
names(dat)<-names(edge)
edge<-rbind(edge,dat)
rm(dat);rm(x)

setdiff(edge$V2,edge$V1); setdiff(edge$V1,edge$V2)
```

```
## integer(0)
```

```
## integer(0)
```

```
#sparseMatrix doesn't like zeroes
edge$V1<-edge$V1+1; edge$V2<-edge$V2+1
edge$V3<-1

#sparse matrix with 1's that designate edges
A <- sparseMatrix(i = edge$V1, j = edge$V2, x = edge$V3)
#stochastic matrix row sum = 1
A <- A/rowSums(A)
all(round(rowSums(A),12) == 1)
```

```
## [1] TRUE
```

```
max(edge);dim(A)
```

```
## [1] 325729
```

```
## [1] 325729 325729
```

(b) Find the dominant eigenvector of M^T and determine the most popular pages. Use $p = .15$.

```
myfunction(A,.15)
```

```
##    user  system elapsed
##   18.54    7.97   27.00
## [1] "Iterations: 183"
```

```
dat<- ( data.frame( as.numeric(x) ,  0:(length(x)-1)) );names(dat)<-c("p","page")
print( head( dat[with(dat, order(p,decreasing = T)), ] , 10) , row.names = F);sum(as.numeric(x))
```

```
##            p   page
##  0.008116197 124802
##  0.006831363  12129
##  0.006032301  83606
##  0.003442274  12838
##  0.003412962  81878
##  0.002069079   1963
##  0.002066350      0
##  0.001881801  10336
##  0.001438036 212843
##  0.001331105  32826
```

```
## [1] 1
```