# DA 605 - Assignment 10

*Dan Fanelli*

```r
library(igraph)
```

## Playing with PageRank

You'll verify for yourself that PageRank works by performing calculations on a small universe of web pages.

Let's use the 6 page universe that we had in the course notes. For this directed graph, perform the following calculations in R.

```r
A <- matrix(c(0,(1/2),(1/2),0,0,0,
              0,0,1,0,0,0,
              (1/4),(1/4),0,0,(1/4),(1/4),
              0,0,0,0,(1/2),(1/2),
              0,0,0,(1/2),0,(1/2),
              0,0,(1/2),(1/2),0,0), nrow=6, ncol=6)
A
```

**Form the A matrix. Then, introduce decay and form the B matrix as we did in the course notes.**

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  0.0    0 0.25  0.0  0.0  0.0
## [2,]  0.5    0 0.25  0.0  0.0  0.0
## [3,]  0.5    1 0.00  0.0  0.0  0.5
## [4,]  0.0    0 0.00  0.0  0.5  0.5
## [5,]  0.0    0 0.25  0.5  0.0  0.0
## [6,]  0.0    0 0.25  0.5  0.5  0.0
```

```r
damping_pct <- 0.15
do_decay <- function(mtrx_in){
  return ((1-damping_pct) * mtrx_in + (damping_pct / nrow(mtrx_in)))
}

B <- do_decay(A)
B
```

```
##       [,1]  [,2]   [,3]  [,4]  [,5]  [,6]
## [1,] 0.025 0.025 0.2375 0.025 0.025 0.025
## [2,] 0.450 0.025 0.2375 0.025 0.025 0.025
## [3,] 0.450 0.875 0.0250 0.025 0.025 0.450
## [4,] 0.025 0.025 0.0250 0.025 0.450 0.450
## [5,] 0.025 0.025 0.2375 0.450 0.025 0.025
## [6,] 0.025 0.025 0.2375 0.450 0.450 0.025
```

```
vector_of_ones <- c(1,1,1,1,1,1)
uniform_rank_vector_r <- vector_of_ones / length(vector_of_ones)
uniform_rank_vector_r
```

**Start with a uniform rank vector r and perform power iterations on B till convergence. That is, compute the solution r = Bn * r. Attempt this for a sufficiently large n so that r actually converges.**

```
## [1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
```

```
power_iterate <- function(mtrx_in, rank_vector, iteration_number){
  cat(sprintf("power_iterate: %s\n", iteration_number))
  rank_vector <- mtrx_in %*% rank_vector
  if(all(rank_vector == (mtrx_in %*% rank_vector))){
    cat(sprintf("YOUR STABLE RANK IS: %s\n", rank_vector))
    return (mtrx_in)
  }else{
    return (power_iterate(mtrx_in, (mtrx_in %*% rank_vector), (iteration_number+1)))
  }
}

power_iterate(B, uniform_rank_vector_r, 0)
```

```
## power_iterate: 0
## power_iterate: 1
## power_iterate: 2
## power_iterate: 3
## power_iterate: 4
## power_iterate: 5
## power_iterate: 6
## power_iterate: 7
## power_iterate: 8
## power_iterate: 9
## power_iterate: 10
## power_iterate: 11
## power_iterate: 12
## power_iterate: 13
## power_iterate: 14
## power_iterate: 15
## power_iterate: 16
## power_iterate: 17
## power_iterate: 18
## power_iterate: 19
## power_iterate: 20
## power_iterate: 21
## power_iterate: 22
## power_iterate: 23
## power_iterate: 24
## power_iterate: 25
## power_iterate: 26
## power_iterate: 27
## YOUR STABLE RANK IS: 0.0773588617577133
##  YOUR STABLE RANK IS: 0.110236378004742
##  YOUR STABLE RANK IS: 0.24639464356571
##  YOUR STABLE RANK IS: 0.186353894445986
##  YOUR STABLE RANK IS: 0.156559266897257
##  YOUR STABLE RANK IS: 0.223096955328592
```

```
##       [,1]  [,2]   [,3]  [,4]  [,5]  [,6]
## [1,] 0.025 0.025 0.2375 0.025 0.025 0.025
## [2,] 0.450 0.025 0.2375 0.025 0.025 0.025
## [3,] 0.450 0.875 0.0250 0.025 0.025 0.450
## [4,] 0.025 0.025 0.0250 0.025 0.450 0.450
## [5,] 0.025 0.025 0.2375 0.450 0.025 0.025
## [6,] 0.025 0.025 0.2375 0.450 0.450 0.025
```

```r
get_unit_vector <- function(vec){
  return (vec / sqrt(sum(vec^2)))
}

eigen_vals <- Re(eigen(B)$values)
# The eigen values are sorted from greatest to least, so:
# http://stackoverflow.com/questions/16616923/find-the-biggest-eigenvalue-in-r
eigen_vals[1]
```

**Compute the eigen-decomposition of B and verify that you indeed get an eigenvalue of 1 as the largest eigenvalue and that its corresponding eigenvector is the same vector that you obtained in the previous power iteration method. Further, this eigenvector has all positive entries and it sums to 1.**

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

```r
eigen_vecs <- Re(eigen(B)$vectors[,1])
eigen_vecs[1]
```

```
## [1] -0.1784825
```

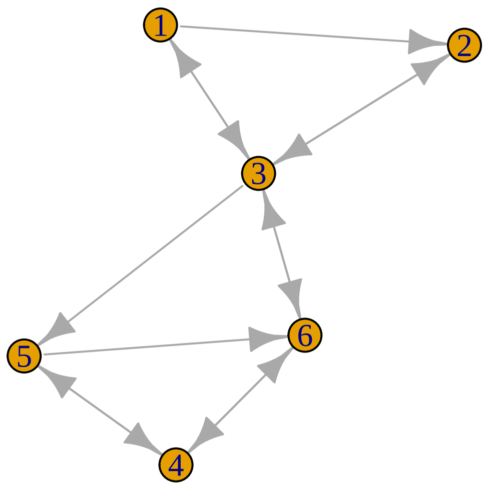
```r
rank_vector_from_printout <- c(0.0773588617577133,0.110236378004742,0.24639464356571,0.186353894445986,

rank_vector_from_printout / eigen_vecs[1] * eigen_vecs[1]
```

```
## [1] 0.07735886 0.11023638 0.24639464 0.18635389 0.15655927 0.22309696
```

```r
# http://www.r-bloggers.com/going-viral-with-rs-igraph-package/
G <- graph(c(1,2,2,3,3,2,1,3,3,1,3,6,6,3,3,5,5,6,6,4,4,6,5,4,4,5), directed = TRUE )
plot(G)
```

**Use the graph package in R and its page.rank method to compute the Page Rank of the graph as given in A. Note that you don't need to apply decay. The package starts with a connected graph and applies decay internally. Verify that you do get the same PageRank vector as the two ap-**



**proaches above**

```
page.rank(G)
```

```
## $vector
## [1] 0.07735886 0.11023638 0.24639464 0.18635389 0.15655927 0.22309696
##
## $value
## [1] 1
##
## $options
## NULL
```

**AND YES IT MATCHES!!!!**