Markov Chains Demonstration Using PageRank

Giancarlo Dibonaventi, Haroon Janjua, Hope Foster-Reyes, Mallika Thanky, Ethan Ton

May 28, 2016

This example demonstrates Markov Chains using the PageRank algorithm.

```
load.graph <- function(graph.file) {
  # Loads graph used by other functions in demo. Assumes file is in working directory.
  #
  # Arguments:
  #   graph.file: Name of xlsx file containing matrix of a graph of interlinked web pages.

  file.data <- read.csv(graph.file)
  data.matrix(file.data) # Convert data to matrix, transpose, and return
}


check.markov <- function(graph) {
  # Checks that the sum of every column = 1.
  #   If it adds to 1, return the graph unchanged.
  #   If it adds to 0 (dangling node), return an adjusted graph, create each value
  #     as 1/nx where nx is total number of nodes.
  #   If it does not add to 1 and is not a dangling node, return FALSE. This represents
  #     an error in the matrix itself.
  #
  # Arguments:
  #   graph: Matrix of a graph of interlinked web pages

  nx <- nrow(graph)

  # Loop through columns and normalize.
  adjusted <- FALSE
  for (i in 1:nx) {
    colsum <- sum(graph[,i])
    if (!isTRUE(all.equal(colsum,1, tolerance=0.0001))) {
      if (colsum == 0) {
        graph[,i] <- 1/nx
        adjusted <- TRUE
      } else {
        return(FALSE)
      }
    }
  }

  if (adjusted) {
    message("Graph adjusted to correct dangling nodes:")
    print(graph)
  }

  return(graph)
}
```

```r
markov.demo <- function(graph, initial, random.factor=0.85, print.skip=3) {
  # Demonstrates iterations of Markov Chain using PageRank algorithm
  #
  # Arguments:
  #   graph: Matrix of a graph of interlinked web pages, forming the transition matrix
  #     representing the probability of state change from j to i, i.e. the probability
  #     of a hypothetical web surfer following a link from the jth page to the ith page.
  #   initial: Initial probability vector.
  #   random.factor: Damping constant simulates random walk accounting for isolated pages.
  #     As written, this factor is the probability that a random surfer will *not*
  #     make a jump to a random page but will follow links.
  #     Set random.factor to 1 to simulate basic Markov Chain without damping.
  #   print.skip: Skip count when printing graphs to demonstrate iterations.

  nx <- nrow(graph) # number of nodes/pages
  probability <- initial

  message("Graph input, representing original transition matrix:")
  print(graph)

  # Check if truly Markov, if not, change problem columns to sum to 1
  graph <- check.markov(graph)
  if (graph[1] == FALSE & length(graph) == 1) {
    stop("ERROR: Data is not properly formatted.")
  }

  # Minimum difference between iteration probability values
  delta_threshold <- 1e-7

  # Iterate until PageRank probability vector is stable to threshold delta, or max 1000 iterations
  for (i in 1:1000) {
    previous <- probability

    # PageRank formula
    probability <- (1 - random.factor) / nx + random.factor * (graph %*% probability)

    # Print alternate iterations.
    if (i %in% 1:5 | (i %% print.skip == 0)) {
      message("Iteration ", i, " PageRank (probability) vector: ")
      print(probability)
    }

    # Check difference between probability and previous probability iteration.
    check_vector <- abs(previous - probability)

    # If all values in check_vector are less than delta_threshold, print result and end.
    if (all(check_vector < delta_threshold)) {
      message("Probabilities converge to steady state vector at iteration number ", i, ": ")
      return(probability)
      break
    }
  }
```

```
  message("Did not reach steady state within 1000 iterations.")
  print(probability)
}


eigen.demo <- function(graph, random.factor=0.85) {

  nx <- length(graph)

  # Check if truly Markov, if not, change problem columns to sum to 1
  #graph <- check.markov(graph)

  # Create Random Walk Matrix (B)
  B <- matrix(1/nx,nrow=nx,ncol=nx)

  # Create PageRank Matrix based off Transition Matrix (graph) and Random Walk Matrix (B)
  M <- (random.factor * graph) + ((1 - random.factor) * B)

  # Create Eigen Vector from the first vector output and change typeof to double (by default, it is com
  eigen_vector <- as.double(eigen(M)$vectors[,1])

  # Normalize vector such that entire column sum = 1
  steady_state_vector <- eigen_vector / sum(eigen_vector)

  # Run check to see if Steady State Vector actually sums to 1
  check <- sum(steady_state_vector)

  if (check == c(1)) {
    print(steady_state_vector)
  } else {
    print("WARNING: Steady State Vector DOES NOT sum to 1")
    print(steady_state_vector)
  }
}
```

Let's start with a simple example. This example confirms that if every page points equally to every other page, the PageRank will be evenly distributed.

Our first output is the graph itself, followed by the Markov iterations which calculate PageRank and gradually reach a steady state.

```
graph <- load.graph("graph-massive-ball.csv")
nx <- nrow(graph)
initial <- rep(1 / nx, nx)
markov.demo(graph, initial, print.skip = 5)
```

```
## Graph input, representing original transition matrix:

##         A   B   C   D   E   F   G   H   I   J
##  [1,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##  [2,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##  [3,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##  [4,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
```

```
##  [5,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##  [6,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##  [7,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##  [8,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##  [9,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
## [10,] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
```

```
## Iteration 1 PageRank (probability) vector:
```

```
##       [,1]
##  [1,]  0.1
##  [2,]  0.1
##  [3,]  0.1
##  [4,]  0.1
##  [5,]  0.1
##  [6,]  0.1
##  [7,]  0.1
##  [8,]  0.1
##  [9,]  0.1
## [10,]  0.1
```

```
## Probabilities converge to steady state vector at iteration number 1:
```

```
##       [,1]
##  [1,]  0.1
##  [2,]  0.1
##  [3,]  0.1
##  [4,]  0.1
##  [5,]  0.1
##  [6,]  0.1
##  [7,]  0.1
##  [8,]  0.1
##  [9,]  0.1
## [10,]  0.1
```

```
#eigen.demo(graph)
```

This example represents a "single hub". Note that all pages point to one hub.

```
graph <- load.graph("graph-single-hub.csv")
markov.demo(graph, initial, print.skip = 5)
```

```
## Graph input, representing original transition matrix:
```

```
##       A B C D E F G H I J
##  [1,] 1 1 1 1 1 1 1 1 1 1
##  [2,] 0 0 0 0 0 0 0 0 0 0
##  [3,] 0 0 0 0 0 0 0 0 0 0
##  [4,] 0 0 0 0 0 0 0 0 0 0
##  [5,] 0 0 0 0 0 0 0 0 0 0
##  [6,] 0 0 0 0 0 0 0 0 0 0
```

```
##  [7,] 0 0 0 0 0 0 0 0 0 0
##  [8,] 0 0 0 0 0 0 0 0 0 0
##  [9,] 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0

## Iteration 1 PageRank (probability) vector:

##        [,1]
##  [1,] 0.865
##  [2,] 0.015
##  [3,] 0.015
##  [4,] 0.015
##  [5,] 0.015
##  [6,] 0.015
##  [7,] 0.015
##  [8,] 0.015
##  [9,] 0.015
## [10,] 0.015

## Iteration 2 PageRank (probability) vector:

##        [,1]
##  [1,] 0.865
##  [2,] 0.015
##  [3,] 0.015
##  [4,] 0.015
##  [5,] 0.015
##  [6,] 0.015
##  [7,] 0.015
##  [8,] 0.015
##  [9,] 0.015
## [10,] 0.015

## Probabilities converge to steady state vector at iteration number 2:

##        [,1]
##  [1,] 0.865
##  [2,] 0.015
##  [3,] 0.015
##  [4,] 0.015
##  [5,] 0.015
##  [6,] 0.015
##  [7,] 0.015
##  [8,] 0.015
##  [9,] 0.015
## [10,] 0.015
```

```
#eigen.demo(graph)
```

What happens if we vary the single hub example so that the hub points only to another site instead of itself? As shown, the hub has the largest PageRank, but the page it points to has nearly as much as the hub "passes" its PageRank to the other site.

```r
graph <- load.graph("graph-hub-transfer.csv")
markov.demo(graph, initial, print.skip = 40)
```

```
## Graph input, representing original transition matrix:

##       A B C D E F G H I J
##  [1,] 0 1 1 1 1 1 1 1 1 1
##  [2,] 1 0 0 0 0 0 0 0 0 0
##  [3,] 0 0 0 0 0 0 0 0 0 0
##  [4,] 0 0 0 0 0 0 0 0 0 0
##  [5,] 0 0 0 0 0 0 0 0 0 0
##  [6,] 0 0 0 0 0 0 0 0 0 0
##  [7,] 0 0 0 0 0 0 0 0 0 0
##  [8,] 0 0 0 0 0 0 0 0 0 0
##  [9,] 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0

## Iteration 1 PageRank (probability) vector:

##        [,1]
##  [1,] 0.780
##  [2,] 0.100
##  [3,] 0.015
##  [4,] 0.015
##  [5,] 0.015
##  [6,] 0.015
##  [7,] 0.015
##  [8,] 0.015
##  [9,] 0.015
## [10,] 0.015

## Iteration 2 PageRank (probability) vector:

##        [,1]
##  [1,] 0.202
##  [2,] 0.678
##  [3,] 0.015
##  [4,] 0.015
##  [5,] 0.015
##  [6,] 0.015
##  [7,] 0.015
##  [8,] 0.015
##  [9,] 0.015
## [10,] 0.015

## Iteration 3 PageRank (probability) vector:

##         [,1]
##  [1,] 0.6933
##  [2,] 0.1867
##  [3,] 0.0150
```

```
##  [4,] 0.0150
##  [5,] 0.0150
##  [6,] 0.0150
##  [7,] 0.0150
##  [8,] 0.0150
##  [9,] 0.0150
## [10,] 0.0150
```

```
## Iteration 4 PageRank (probability) vector:
```

```
##           [,1]
##  [1,] 0.275695
##  [2,] 0.604305
##  [3,] 0.015000
##  [4,] 0.015000
##  [5,] 0.015000
##  [6,] 0.015000
##  [7,] 0.015000
##  [8,] 0.015000
##  [9,] 0.015000
## [10,] 0.015000
```

```
## Iteration 5 PageRank (probability) vector:
```

```
##            [,1]
##  [1,] 0.6306593
##  [2,] 0.2493408
##  [3,] 0.0150000
##  [4,] 0.0150000
##  [5,] 0.0150000
##  [6,] 0.0150000
##  [7,] 0.0150000
##  [8,] 0.0150000
##  [9,] 0.0150000
## [10,] 0.0150000
```

```
## Iteration 40 PageRank (probability) vector:
```

```
##            [,1]
##  [1,] 0.4670154
##  [2,] 0.4129846
##  [3,] 0.0150000
##  [4,] 0.0150000
##  [5,] 0.0150000
##  [6,] 0.0150000
##  [7,] 0.0150000
##  [8,] 0.0150000
##  [9,] 0.0150000
## [10,] 0.0150000
```

```
## Iteration 80 PageRank (probability) vector:
```

```
##            [,1]
##  [1,] 0.4675667
##  [2,] 0.4124333
##  [3,] 0.0150000
##  [4,] 0.0150000
##  [5,] 0.0150000
##  [6,] 0.0150000
##  [7,] 0.0150000
##  [8,] 0.0150000
##  [9,] 0.0150000
## [10,] 0.0150000

## Probabilities converge to steady state vector at iteration number 98:

##            [,1]
##  [1,] 0.4675675
##  [2,] 0.4124325
##  [3,] 0.0150000
##  [4,] 0.0150000
##  [5,] 0.0150000
##  [6,] 0.0150000
##  [7,] 0.0150000
##  [8,] 0.0150000
##  [9,] 0.0150000
## [10,] 0.0150000
```

```
#eigen.demo(graph)
```

This example is from the Cornell lecture by Raluca Remus.

```
graph <- load.graph("graph-remus.csv")
nx <- nrow(graph)
initial <- rep(1 / nx, nx)
markov.demo(graph, initial, print.skip = 20)
```

```
## Graph input, representing original transition matrix:

##              A   B C   D
## [1,] 0.0000000 0.0 1 0.5
## [2,] 0.3333333 0.0 0 0.0
## [3,] 0.3333333 0.5 0 0.5
## [4,] 0.3333333 0.5 0 0.0

## Iteration 1 PageRank (probability) vector:

##           [,1]
## [1,] 0.3562500
## [2,] 0.1083333
## [3,] 0.3208333
## [4,] 0.2145833

## Iteration 2 PageRank (probability) vector:
```

```
##           [,1]
## [1,] 0.4014062
## [2,] 0.1384375
## [3,] 0.2756771
## [4,] 0.1844792
```

```
## Iteration 3 PageRank (probability) vector:
```

```
##           [,1]
## [1,] 0.3502292
## [2,] 0.1512318
## [3,] 0.2884714
## [4,] 0.2100677
```

```
## Iteration 4 PageRank (probability) vector:
```

```
##           [,1]
## [1,] 0.3719794
## [2,] 0.1367316
## [3,] 0.2902839
## [4,] 0.2010051
```

```
## Iteration 5 PageRank (probability) vector:
```

```
##           [,1]
## [1,] 0.3696685
## [2,] 0.1428942
## [3,] 0.2864323
## [4,] 0.2010051
```

```
## Iteration 20 PageRank (probability) vector:
```

```
##           [,1]
## [1,] 0.3681507
## [2,] 0.1418094
## [3,] 0.2879616
## [4,] 0.2020783
```

```
## Probabilities converge to steady state vector at iteration number 20:
```

```
##           [,1]
## [1,] 0.3681507
## [2,] 0.1418094
## [3,] 0.2879616
## [4,] 0.2020783
```

```
#eigen.demo(graph)
```

This more complex example demonstrates two hubs.

```r
graph <- load.graph("graph-dual-hub.csv")
nx <- nrow(graph)
initial <- rep(1 / nx, nx)
markov.demo(graph, initial, print.skip = 20)
```

```
## Graph input, representing original transition matrix:

##       A B C   D   E   F   G   H   I J K
##  [1,] 0 0 0 0.5 0.0 0.0 0.0 0.0 0.0 0 0
##  [2,] 0 0 1 0.5 0.5 0.5 0.5 0.5 0.5 0 0
##  [3,] 0 1 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [4,] 0 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [5,] 0 0 0 0.0 0.0 0.5 0.5 0.5 0.5 1 1
##  [6,] 0 0 0 0.0 0.5 0.0 0.0 0.0 0.0 0 0
##  [7,] 0 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [8,] 0 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [9,] 0 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
## [10,] 0 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
## [11,] 0 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0

## Graph adjusted to correct dangling nodes:

##                A B C   D   E   F   G   H   I J K
##  [1,] 0.09090909 0 0 0.5 0.0 0.0 0.0 0.0 0.0 0 0
##  [2,] 0.09090909 0 1 0.5 0.5 0.5 0.5 0.5 0.5 0 0
##  [3,] 0.09090909 1 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [4,] 0.09090909 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [5,] 0.09090909 0 0 0.0 0.0 0.5 0.5 0.5 0.5 1 1
##  [6,] 0.09090909 0 0 0.0 0.5 0.0 0.0 0.0 0.0 0 0
##  [7,] 0.09090909 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [8,] 0.09090909 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
##  [9,] 0.09090909 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
## [10,] 0.09090909 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
## [11,] 0.09090909 0 0 0.0 0.0 0.0 0.0 0.0 0.0 0 0

## Iteration 1 PageRank (probability) vector:

##             [,1]
##  [1,] 0.05929752
##  [2,] 0.32975207
##  [3,] 0.09793388
##  [4,] 0.02066116
##  [5,] 0.32975207
##  [6,] 0.05929752
##  [7,] 0.02066116
##  [8,] 0.02066116
##  [9,] 0.02066116
## [10,] 0.02066116
## [11,] 0.02066116

## Iteration 2 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02699944
##  [2,] 0.30193229
##  [3,] 0.29850770
##  [4,] 0.01821844
##  [5,] 0.10488683
##  [6,] 0.15836307
##  [7,] 0.01821844
##  [8,] 0.01821844
##  [9,] 0.01821844
## [10,] 0.01821844
## [11,] 0.01821844
```

```
## Iteration 3 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02346552
##  [2,] 0.41230680
##  [3,] 0.27236513
##  [4,] 0.01572268
##  [5,] 0.13722686
##  [6,] 0.06029959
##  [7,] 0.01572268
##  [8,] 0.01572268
##  [9,] 0.01572268
## [10,] 0.01572268
## [11,] 0.01572268
```

```
## Iteration 4 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02213175
##  [2,] 0.35763727
##  [3,] 0.36591038
##  [4,] 0.01544961
##  [5,] 0.08785192
##  [6,] 0.07377103
##  [7,] 0.01544961
##  [8,] 0.01544961
##  [9,] 0.01544961
## [10,] 0.01544961
## [11,] 0.01544961
```

```
## Iteration 5 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02191263
##  [2,] 0.42132446
##  [3,] 0.31933823
##  [4,] 0.01534654
##  [5,] 0.09266182
##  [6,] 0.05268361
```

```
##  [7,] 0.01534654
##  [8,] 0.01534654
##  [9,] 0.01534654
## [10,] 0.01534654
## [11,] 0.01534654
```

```
## Iteration 20 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02183629
##  [2,] 0.39676595
##  [3,] 0.35687201
##  [4,] 0.01532371
##  [5,] 0.08228753
##  [6,] 0.05029594
##  [7,] 0.01532371
##  [8,] 0.01532371
##  [9,] 0.01532371
## [10,] 0.01532371
## [11,] 0.01532371
```

```
## Iteration 40 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02183629
##  [2,] 0.39899877
##  [3,] 0.35463923
##  [4,] 0.01532371
##  [5,] 0.08228752
##  [6,] 0.05029591
##  [7,] 0.01532371
##  [8,] 0.01532371
##  [9,] 0.01532371
## [10,] 0.01532371
## [11,] 0.01532371
```

```
## Iteration 60 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02183629
##  [2,] 0.39908531
##  [3,] 0.35455268
##  [4,] 0.01532371
##  [5,] 0.08228752
##  [6,] 0.05029591
##  [7,] 0.01532371
##  [8,] 0.01532371
##  [9,] 0.01532371
## [10,] 0.01532371
## [11,] 0.01532371
```

```
## Iteration 80 PageRank (probability) vector:
```

```
##             [,1]
##  [1,] 0.02183629
##  [2,] 0.39908867
##  [3,] 0.35454933
##  [4,] 0.01532371
##  [5,] 0.08228752
##  [6,] 0.05029591
##  [7,] 0.01532371
##  [8,] 0.01532371
##  [9,] 0.01532371
## [10,] 0.01532371
## [11,] 0.01532371
```

```
## Probabilities converge to steady state vector at iteration number 87:
```

```
##             [,1]
##  [1,] 0.02183629
##  [2,] 0.39908884
##  [3,] 0.35454915
##  [4,] 0.01532371
##  [5,] 0.08228752
##  [6,] 0.05029591
##  [7,] 0.01532371
##  [8,] 0.01532371
##  [9,] 0.01532371
## [10,] 0.01532371
## [11,] 0.01532371
```

```
#eigen.demo(graph)
```