Markov Chains Demonstration Using Page Rank

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
}

markov.demo <- function(graph, 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.
  #   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.
  #   print.skip: Skip count when printing graphs to demonstrate iterations.

  # initial probability vector
  nx <- nrow(graph) # number of nodes/pages
  initial <- rep(1 / nx, nx)
  # initial <- c(1, rep(0, nx - 1))
  probability <- initial

  # Minimum difference between iteration probability values
  delta_threshold <- 1e-10

  i <- 1
  print(graph)

  # Iterate until PageRank probability vector is stable to threshold delta
  repeat{
    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)) {
      cat("Iteration", i, ": ")
      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)) {
      cat("Probabilities converge to steady state vector at iteration number", i, ": ")
      print(probability)
      break
    } else if (i == 1000) {
      cat("Did not reach steady state within 1000 iterations")
      print(probability)
      break
    }

    i <- i + 1
  }
}
```

Let's start with a simple example. Our first output is the graph itself, followed by the Markov iterations which calculate PageRank and gradually reach a steady state.

```
markov.demo(load.graph("graph-simple.csv"), print.skip = 5)
```

```
##         A   B C         D
## [1,] 0.25 0.5 1 0.3333333
## [2,] 0.25 0.0 0 0.3333333
## [3,] 0.25 0.5 0 0.3333333
## [4,] 0.25 0.0 0 0.0000000
## Iteration 1 :           [,1]
## [1,] 0.4802083
## [2,] 0.1614583
## [3,] 0.2677083
## [4,] 0.0906250
## Iteration 2 :           [,1]
## [1,] 0.4613932
## [2,] 0.1652214
## [3,] 0.2338411
## [4,] 0.1395443
## Iteration 3 :           [,1]
## [1,] 0.4440677
## [2,] 0.1750836
## [3,] 0.2453027
## [4,] 0.1355461
## Iteration 4 :           [,1]
## [1,] 0.4531869
## [2,] 0.1702691
## [3,] 0.2446796
## [4,] 0.1318644
## Iteration 5 :           [,1]
## [1,] 0.4515058
## [2,] 0.1711638
```

```
## [3,] 0.2435282
## [4,] 0.1338022
## Iteration 10 :           [,1]
## [1,] 0.4513809
## [2,] 0.1712168
## [3,] 0.2439866
## [4,] 0.1334158
## Iteration 15 :           [,1]
## [1,] 0.4513763
## [2,] 0.1712191
## [3,] 0.2439872
## [4,] 0.1334174
## Iteration 20 :           [,1]
## [1,] 0.4513763
## [2,] 0.1712191
## [3,] 0.2439872
## [4,] 0.1334175
## Probabilities converge to steady state vector at iteration number 22 :           [,1]
## [1,] 0.4513763
## [2,] 0.1712191
## [3,] 0.2439872
## [4,] 0.1334175
```

This example is from the Cornell lecture by Raluca Remus.

```
markov.demo(load.graph("graph-remus.csv"), print.skip = 20)
```

```
##         A   B C   D
## [1,] 0.00 0.0 1 0.5
## [2,] 0.33 0.0 0 0.0
## [3,] 0.33 0.5 0 0.5
## [4,] 0.33 0.5 0 0.0
## Iteration 1 :          [,1]
## [1,] 0.356250
## [2,] 0.107625
## [3,] 0.320125
## [4,] 0.213875
## Iteration 2 :           [,1]
## [1,] 0.4005031
## [2,] 0.1374281
## [3,] 0.2740656
## [4,] 0.1831687
## Iteration 3 :           [,1]
## [1,] 0.3483025
## [2,] 0.1498411
## [3,] 0.2860948
## [4,] 0.2082481
## Iteration 4 :           [,1]
## [1,] 0.3691860
## [2,] 0.1351989
## [3,] 0.2873868
## [4,] 0.1988813
## Iteration 5 :           [,1]
```

```
## [1,] 0.3663033
## [2,] 0.1410567
## [3,] 0.2830408
## [4,] 0.1985162
## Iteration 20 :           [,1]
## [1,] 0.3614207
## [2,] 0.1388937
## [3,] 0.2820635
## [4,] 0.1979313
## Iteration 40 :           [,1]
## [1,] 0.3611315
## [2,] 0.1387979
## [3,] 0.2818474
## [4,] 0.1977873
## Iteration 60 :           [,1]
## [1,] 0.3611212
## [2,] 0.1387945
## [3,] 0.2818396
## [4,] 0.1977822
## Iteration 80 :           [,1]
## [1,] 0.3611208
## [2,] 0.1387944
## [3,] 0.2818393
## [4,] 0.1977820
## Iteration 100 :           [,1]
## [1,] 0.3611208
## [2,] 0.1387944
## [3,] 0.2818393
## [4,] 0.1977820
## Probabilities converge to steady state vector at iteration number 100 :           [,1]
## [1,] 0.3611208
## [2,] 0.1387944
## [3,] 0.2818393
## [4,] 0.1977820
```

This example confirms that if every page points equally to every other page, the PageRank will be evenly distributed.

```
markov.demo(load.graph("graph-massive-ball.csv"), print.skip = 5)
```

```
##         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 :       [,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
```

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

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

```
##       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 :        [,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 :        [,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
```

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.

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

```
##       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 :        [,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 :        [,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 :         [,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 :           [,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 :            [,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 :            [,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 :            [,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
## Iteration 120 :            [,1]
##  [1,] 0.4675676
##  [2,] 0.4124324
##  [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 141 :            [,1]
##  [1,] 0.4675676
##  [2,] 0.4124324
##  [3,] 0.0150000
##  [4,] 0.0150000
##  [5,] 0.0150000
##  [6,] 0.0150000
##  [7,] 0.0150000
##  [8,] 0.0150000
##  [9,] 0.0150000
## [10,] 0.0150000
```