Rowan Sharman

Linearity I Final Project

7 May 2017

Markov Chain Analysis

For this project I decided to write a program in Python that analyzes a network flow problem to determine whether it will reach an equilibrium state, then simulates the flow over a user-defined number of steps. In order to more deeply understand the operations being used, I wrote the simulation part of the program using only elementary operations. The end state prediction uses numpy to find the eigenvalues of the transition matrix.

Simulation:

All of the matrix operations in the simulation are written with elementary operations. Here is a brief overview of each of the functions I wrote:

dot: Takes two equal-dimension vectors and sums the products of the elements in the same positions. Returns this sum.

multiply: Takes two matrices and dots the rows of the first one by the columns of the second one. Returns a matrix where the item in column *i*, row *j* is the dot product of the *j*th row in the first matrix and the *i*th column in the second matrix.

exponent: Takes a matrix, *M*, and a power, *p*. Returns the identity matrix multiplied recursively by *M* *p* times. (Returns *I* for *p*=0, *M* for *p*=1, *M*x*M* for *p*=2, etc)

normalize: Divides each element in a matrix by the sum of its column.

When the program starts, it prompts the user to enter the number of nodes in the network, *n*, then helps the user generate the initial state vector. Optionally, it then normalizes the state vector so that the entries sum to one, meaning that the values now represent the percentage at each node in the initial state.

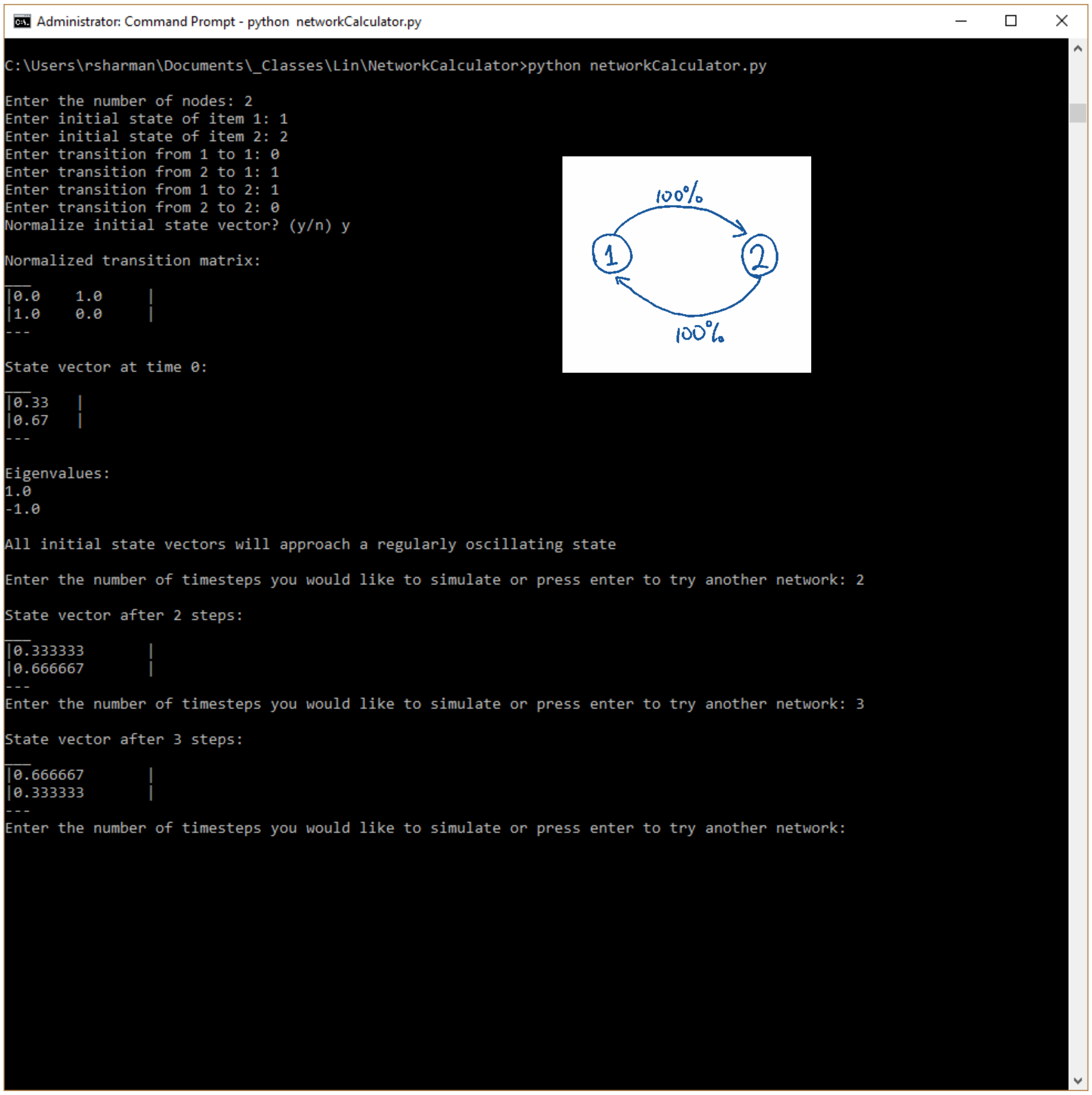
Next, the user is asked to enter the transitions between each node. These transitions are stored in an *n* x *n* matrix where the item in column *i*, row *j* represents the transition from the *j*th node to the *i*th node in the state vector. This matrix is then normalized by columns so that not more than 100 percent is transferred out of each node at each timestep.

The overall transition matrix is found by taking the single-step transition matrix to the power of the number of time steps required.

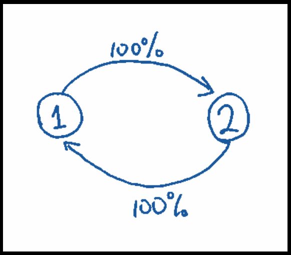
Lastly, the final state vector is computed by multiplying the overall transition matrix by the initial state vector. This vector will still sum to one, or to what the initial state vector summed to if the user chose not to normalize the initial state vector.

End State Prediction:

Before running the simulation, the program can predict some basic information about the final state vector at different timesteps. It does this by finding and analyzing the eigenvalues. If any one of the eigenvalues is -1, the end state will always oscillate between two conditions, because the value of c λ^nv changes sign as n changes. A simple example of a network which exhibits this behavior is shown below.



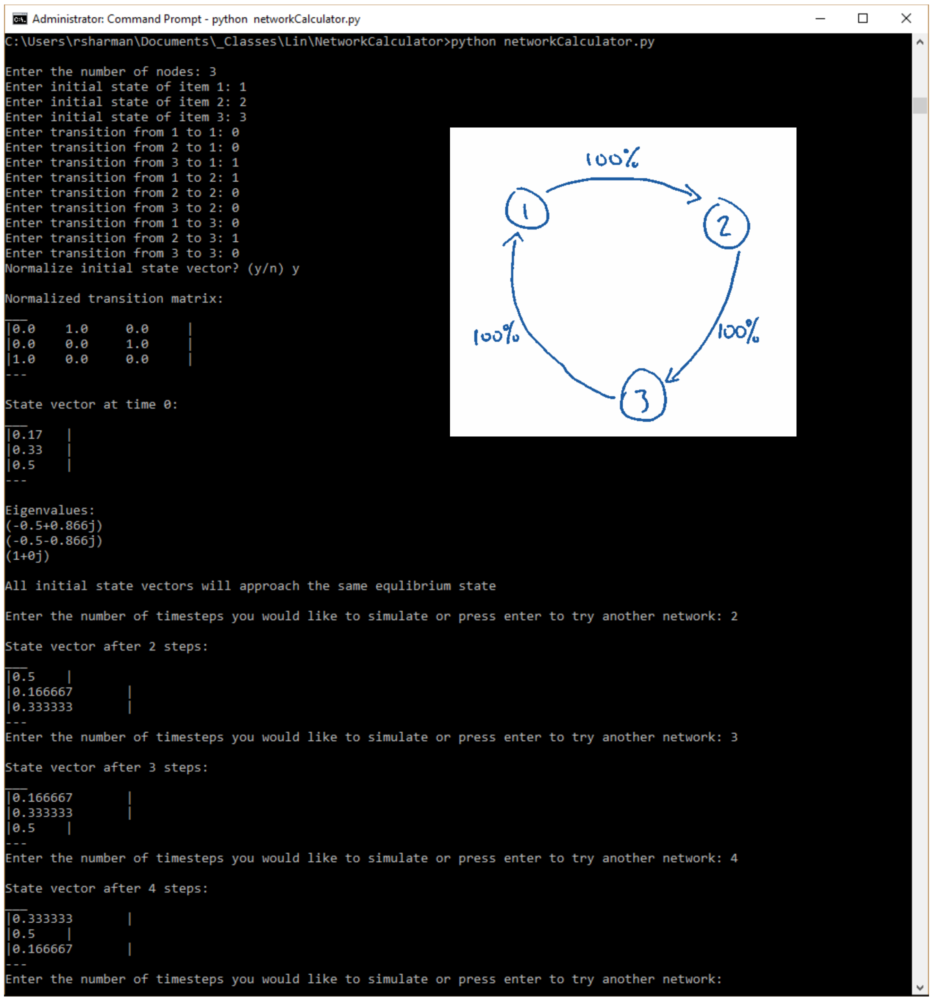
```
C:\Users\rsharman\Documents\_Classes\Lin\NetworkCalculator>python networkCalculator.py

Enter the number of nodes: 2
Enter initial state of item 1: 1
Enter initial state of item 2: 2
Enter transition from 1 to 1: 0
Enter transition from 2 to 1: 1
Enter transition from 1 to 2: 1
Enter transition from 2 to 2: 0
Normalize initial state vector? (y/n) y

Normalized transition matrix:
___
|0.0    1.0     |
|1.0    0.0     |
---

State vector at time 0:
___
|0.33   |
|0.67   |
---

Eigenvalues:
1.0
-1.0

All initial state vectors will approach a regularly oscillating state

Enter the number of timesteps you would like to simulate or press enter to try another network: 2

State vector after 2 steps:
___
|0.333333       |
|0.666667       |
---
Enter the number of timesteps you would like to simulate or press enter to try another network: 3

State vector after 3 steps:
___
|0.666667       |
|0.333333       |
---
Enter the number of timesteps you would like to simulate or press enter to try another network:
```



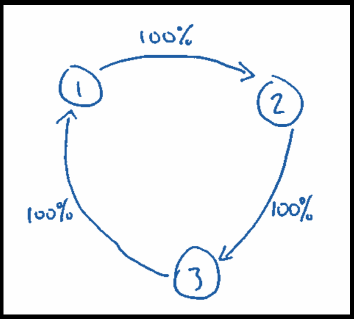
If none of the eigenvalues are -1, there are two other options: if there is only one other eigenvalue that is 1, all initial state vectors will go to the same final state after enough time because the eigenvalue 1 becomes the only significant one after some time.

The other option is that there are two or more eigenvalues of 1. In this case, all initial state vectors will go to equilibrium, but the equilibrium point depends on the initial state vector.

There is one other state which my program does not know how to handle: the case of imaginary eigenvalues, such as the network shown below. As you can see, the network reaches an oscillating state, but it oscillates among three nodes. My program can accurately simulate this network, but the prediction cannot handle imaginary numbers, and incorrectly assesses the outcome state.


Administrator: Command Prompt - python networkCalculator.py

```
C:\Users\rsharman\Documents\_Classes\Lin\NetworkCalculator>python networkCalculator.py

Enter the number of nodes: 3
Enter initial state of item 1: 1
Enter initial state of item 2: 2
Enter initial state of item 3: 3
Enter transition from 1 to 1: 0
Enter transition from 2 to 1: 0
Enter transition from 3 to 1: 1
Enter transition from 1 to 2: 1
Enter transition from 2 to 2: 0
Enter transition from 3 to 2: 0
Enter transition from 1 to 3: 0
Enter transition from 2 to 3: 1
Enter transition from 3 to 3: 0
Normalize initial state vector? (y/n) y

Normalized transition matrix:
___
|0.0    1.0     0.0     |
|0.0    0.0     1.0     |
|1.0    0.0     0.0     |
---

State vector at time 0:
___
|0.17   |
|0.33   |
|0.5    |
---

Eigenvalues:
(-0.5+0.866j)
(-0.5-0.866j)
(1+0j)

All initial state vectors will approach the same equlibrium state

Enter the number of timesteps you would like to simulate or press enter to try another network: 2

State vector after 2 steps:
___
|0.5    |
|0.166667       |
|0.333333       |
---
Enter the number of timesteps you would like to simulate or press enter to try another network: 3

State vector after 3 steps:
___
|0.166667       |
|0.333333       |
|0.5    |
---
Enter the number of timesteps you would like to simulate or press enter to try another network: 4

State vector after 4 steps:
___
|0.333333       |
|0.5    |
|0.166667       |
---
Enter the number of timesteps you would like to simulate or press enter to try another network:
```


100%
1
2
100%
100%
3