a) state the transition matrix P for MC(a)

```
import math
import numpy as np

#main
Pa = np.matrix(
(
(0,0.5,0.5,0,0,0,),
(1,0,0,0,0,0),
(0,0.9,0,0.1,0,0),
(0,0,0,0,1,0),
(0,0,0,0,0,1),
(0,0,0,1,0,0)
)
)
```

b) irreducibility test over a MC defined by transition matrix P

```
import math
import numpy as np
import networkx as nx


def DFSrec(G,v,visited):
    visited[v]= True

    for i in list(G.neighbors(v)):
        if visited[i]==False:
            DFSrec(G, i, visited)

def DFS(G, startAt):
    visited = [False]*(G.number_of_nodes())
    DFSrec(G,startAt,visited)

    if any(i == False for i in visited):
        return False

    return True

def isStronglyConnected(G):
    if DFS(G,0) == False:
        return False

    # returns a graph with the same set of V and E of G but having all
    # directions inverted
    T = G.reverse(True)
    return DFS(T,0)

def createGraphFromMC(P):
    # Create Directed Graph
    G=nx.DiGraph()

    # Add a list of nodes:
    G.add_nodes_from(range(0,len(P)))

    # Add a list of edges:
```

```
    for i in range(0,len(P)):
        for j in range(0,len(P)):
            if P[i,j]>0:
                G.add_edge(i,j)

    return G

def isIrreducibleMC(P):
    G = createGraphFromMC(P)
    print "irreducible" if isStronglyConnected(G) else "NOT irreducible"
```

thinking of the MC as a directed graph G(V,E), the MC is irreducible if and only if the G is strongly connected. This condition can be assessed in many ways. A longer one consists in applying a DepthFirstVisit of G for all its vertices v $\in$V. If for each v $\in$V the DFS is able to visit other vertices, G is strongly connected.

A shorter approach (Kosaraju) provides an answer applying only two DFS at the same v, the second of which is run after having inverted the edges of G.

c) periodicity test over a MC defined by transition matrix P (to be adjusted possibly)

d) power of transition matrix

```
import math
import numpy as np

#main
Pa = np.matrix(
(
(0,0.5,0.5,0,0,0,),
(1,0,0,0,0,0),
(0,0.9,0,0.1,0,0),
(0,0,0,0,1,0),
(0,0,0,0,0,1),
(0,0,0,1,0,0)
)
)

Pb = np.matrix(
(
(0,0.5,0.5,0,0,0,),
(1,0,0,0,0,0),
(0,0.9,0,0.1,0,0),
(0,0,0,0,1,0),
(0,0,0,0,0,1),
(0.5,0,0,0.5,0,0)
)
)


print Pa**50
```

```
print Pb**50
```

e) invariant distribution π over a MC defined by transition matrix P

```
import math
import numpy as np

# https://stephens999.github.io/fiveMinuteStats/stationary_distribution.html
def findInvDistrib(P):

    I = np.identity(len(P))

    # add condition over pi all entries sum to 1
    newrow = np.ones(len(P))
    A = P
    A = A.transpose()
    A = np.vstack([A-I, newrow])
    print A

    B = np.zeros(len(P)+1)
    B[len(P)] = 1

    x, residuals, rank, s = np.linalg.lstsq(A, B)
    print "Solution: ", x
    print "Residuals: ", residuals
    print "Rank: ", rank
    print "Test1: ", x*P
    print "Test2: ", P**1000



'''
    try:
        x = np.linalg.solve(A, B)
        print 'Invariant distribution found'
        print x
    except np.linalg.LinAlgError as err:
        if 'Singular matrix' in str(err):
            print 'Singular matrix passed'
        else:
            raise
'''

#main
Pa = np.matrix(
(
(0,0.5,0.5,0,0,0,),
(1,0,0,0,0,0),
(0,0.9,0,0.1,0,0),
(0,0,0,0,1,0),
(0,0,0,0,0,1),
(0,0,0,1,0,0)
)
)
```

```
Pb = np.matrix(
(
(0,0.5,0.5,0,0,0,),
(1,0,0,0,0,0),
(0,0.9,0,0.1,0,0),
(0,0,0,0,1,0),
(0,0,0,0,0,1),
(0.5,0,0,0.5,0,0)
)
)

findInvDistrib(Pa)
findInvDistrib(Pb)
```

A system of equations can be set to solve $\pi P=\pi$ adding the following constraint over π:
$\pi 1 + \pi 2 + \dots \pi n = 1$

Since the matrix defining the system is no longer square Python linalg.solve cannot be applied. We can use linalg.lstsq anyway . If residuals are vanishingly small we can assume the solution is unique and compare the result with conditions over the MC required for uniqueness of π to hold (any irreducible and aperiodic MC on a finite state space as a unique steady distribution).