tmon-p1

October 20, 2023

1 Montecarlo Techniques - Practice 1

Andrés Herencia López-Menchero

MUTECI - 2023/2024

1.1 Old car simulation process

1.1.1 Implement the simulation model of the old car problem in R or python.

[1]:
```
import numpy as np
import random

### Implement the simulation model of the old car problem in R or python.

def dtmc(N = 10**4, transitionMatrix = [[1, 0, 0, 0],
                                        [0.05, 0.50, 0.20, 0.25],
                                        [0.10, 0.75, 0, 0.15],
                                        [0, 0, 0, 1]],
         initialState = 1, finalState = 0):

    """
    Simulate a Discrete-Time Markov Chain (DTMC).

    Parameters:
    - N (int): The number of steps to simulate the Markov Chain. Default is␣
 ↪1000.
    - transitionMatrix (list of lists): The transition matrix representing the␣
 ↪DTMC. Default is a 4x4 matrix.
    - initialState (int): The initial state from which the simulation starts.␣
 ↪Default is state 1.
    - finalState (int): The absorption state, if any, at which the simulation␣
 ↪should stop. Default is state 0.

    Returns:
    - t: A list which contains the number of steps from the initialState to the␣
 ↪finalState in each iteration.
```

```
    - p: A list which indicates whether the iteration has reach to the␣
↪finalState (1) or not (0).
    - states: A list that saves all the states it has gone through since the␣
↪initial state for each iteration.
    """

    # list where we will save our states
    states = [initialState]
    np.random.seed(123)
    p = np.zeros(N)              # final state reached.
    t = np.full(N,np.nan)              # simulation time.
    states = [[] for _ in range(N)]     # states.

    # check if we have an absorption state
    absorption_state = []
    for tm in range(0,len(transitionMatrix)):
        for m in transitionMatrix[tm]:
            if (m == 1)&(transitionMatrix[tm].index(m) == tm):
                absorption_state.append(tm)

    # simulation
    for n in range(N):
        i = initialState
        states[n].append(i)
        while (i not in absorption_state):
            j = -1; # columns of our transition matrix (this is the 'next␣
↪state' column)
            pj = 0; # transition probability from state 'i' to state 'j'.
            threshold = np.random.uniform(0, 1)
            while (pj < threshold):
                j += 1
                pj = pj + transitionMatrix[i][j]
            i = j
            states[n].append(i)
            if i == finalState:
                t[n] = len(states[n])-1
                p[n] += 1
    return(t,p,states)
```

[2]:
```
help(dtmc)
```

```
Help on function dtmc in module __main__:

dtmc(N=10000, transitionMatrix=[[1, 0, 0, 0], [0.05, 0.5, 0.2, 0.25], [0.1,
0.75, 0, 0.15], [0, 0, 0, 1]], initialState=1, finalState=0)
    Simulate a Discrete-Time Markov Chain (DTMC).
```

```
    Parameters:
    - N (int): The number of steps to simulate the Markov Chain. Default is
1000.
    - transitionMatrix (list of lists): The transition matrix representing the
DTMC. Default is a 4x4 matrix.
    - initialState (int): The initial state from which the simulation starts.
Default is state 1.
    - finalState (int): The absorption state, if any, at which the simulation
should stop. Default is state 0.

    Returns:
    - t: A list which contains the number of steps from the initialState to the
finalState in each iteration.
    - p: A list which indicates whether the iteration has reach to the
finalState (1) or not (0).
    - states: A list that saves all the states it has gone through since the
initial state for each iteration.
```

1.1.2 Example of use and some metrics

[3]:
```
N = 10**5
initialState = 1
finalState = 0
transitionMatrix = [[1, 0, 0, 0],
                    [0.05, 0.50, 0.20, 0.25],
                    [0.10, 0.75, 0, 0.15],
                    [0, 0, 0, 1]]
[t,p,s] =␣
 ↪dtmc(N=N,transitionMatrix=transitionMatrix,initialState=initialState,finalState=finalState)
print('The number of steps consumed to reach to the final state in each␣
 ↪iteration is '+str(t))
print('The times that our function has reached to the final state␣
 ↪is',str(sum(p)),'in',N,'simulations')
print('The history of states in',N,'simulations is',str(s[0:5]))
```

```
The number of steps consumed to reach to the final state in each iteration is
[nan 24. nan … nan nan nan]
The times that our function has reached to the final state is 19887.0 in 100000
simulations
The history of states in 100000 simulations is [[1, 2, 1, 1, 2, 1, 1, 3], [1, 2,
1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 2, 0], [1, 1, 1,
1, 1, 1, 1, 3], [1, 3], [1, 1, 2, 1, 1, 1, 3]]
```

1.1.3 Approximate the absorption time from `Operativo` to `Siniestrado` through simulation (years).

[4]:
```
np.nanmean(dtmc(N=N, initialState = 1, finalState = 0)[0])
```

[4]: 3.571227435007794

1.1.4 Approximate through simulation the probability of absorption from `Averiado` to `Vendido` (%).

[5]:
```
round(sum(dtmc(N=N,initialState = 2, finalState = 3)[1])*100/N,4)
```

[5]: 75.168