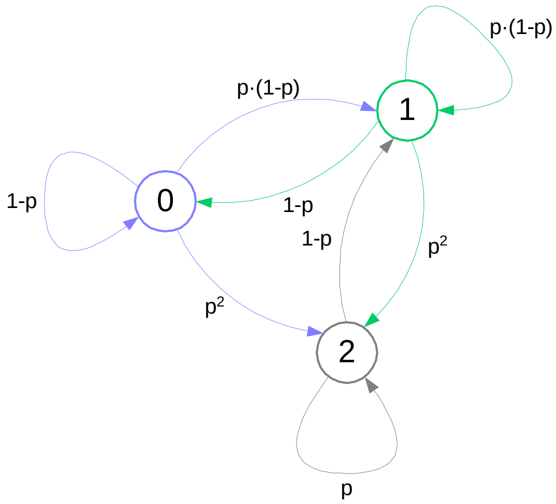
Problem 1

In the tutorial we discussed one modeling problem involving a TH-DTMC:



with transition matrix

$$P = \begin{pmatrix} 1-p & p\cdot(1-p) & p^2 \\ 1-p & p\cdot(1-p) & p^2 \\ 0 & 1-p & p \end{pmatrix}$$

where $p \in (0,1)$ *is a parameter. Show that:*

- *The TH-DTMC is irreducible and aperiodic.*
- *Find the steady-state vector* $\pi = \left(\pi^{(0)}, \pi^{(1)}, \pi^{(2)}\right)$ *(this vector is guaranteed to exist since all finite, irreducible and aperiodic TH-DTMCs are also positive recurrent).*

A DTMC is irreducible if between any two of the states of its state space exists a path that connects them. Obviously, all three states of the given diagram are directly connected except for state 2 → state 0 where a path via state 1 exists.

A state is periodic if the only way to return to itself is through paths of length $k \cdot d$ for some values of k and a fixed value of $d > 1$. One can see on first glance in the diagram above that there are paths for each state returning to themselves. Therefore, we get $d = 1$, a contradiction to the definition of the term "periodic". If a state is not periodic, then it is aperiodic. In addition, the given DTMC is ergodic because it is aperiodic *and* positive recurrent (as defined above).

The steady-state vector ought to fulfill two basic equations:

$$\pi = \pi \cdot P$$

$$1 = \sum_{i=0}^{N} \pi_i$$

The latter (called normalization condition) can be rewritten in our case as:

$$1 = \pi_0 + \pi_1 + \pi_2$$

In the steady state the total flow out of a state is equal to the total flow into that state. Based on this property, called flow balance, one can write flow balance equations for any state of the process.

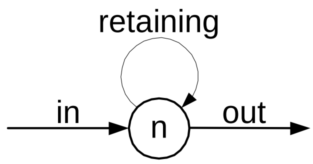


Figure 1: Flow to and out of a state

The flow from a state to itself (retaining) belongs both to the in- and out-flow and can be omitted in order to further simplify the formulas:

$$\sum_{\text{all states } j}\left(in_j \cdot \pi_i\right) = \sum_{\text{all states } j}\left(out_j \cdot \pi_j\right)$$
$$\pi_i \cdot \sum_{\text{all states } j \neq i} in_j = \sum_{\text{all states } j \neq i}\left(out_j \cdot \pi_j\right)$$

However, solving $\pi = \pi \cdot P$ seems to fabricate usually shorter formulas. We can write now:

$$\pi_0 = (1-p)\cdot\pi_0 + (1-p)\cdot\pi_1$$
$$\pi_1 = p\cdot(1-p)\cdot\pi_0 + p\cdot(1-p)\cdot\pi_1 + (1-p)\cdot\pi_2$$
$$\pi_2 = p^2\cdot\pi_0 + p^2\cdot\pi_1 + p\cdot\pi_2$$

First, we express π_0 in terms of π_1:

$$\pi_0 = (1-p)\cdot\pi_0 + (1-p)\cdot\pi_1$$
$$p\cdot\pi_0 = (1-p)\cdot\pi_1$$
$$\pi_0 = \frac{1-p}{p}\cdot\pi_1$$

Now, we repeat the same procedure for π_1 and apply the knowledge gained recently:

$$\pi_1 = p\cdot(1-p)\cdot\pi_0 + p\cdot(1-p)\cdot\pi_1 + (1-p)\cdot\pi_2$$
$$(1-p\cdot(1-p))\cdot\pi_1 = (1-p)^2\cdot\pi_1 + (1-p)\cdot\pi_2$$
$$p\cdot\pi_1 = (1-p)\cdot\pi_2$$
$$\pi_1 = \frac{1-p}{p}\cdot\pi_2$$

The normalization condition gives:

$$1 = \pi_0 + \pi_1 + \pi_2$$

$$1 = \left(\frac{1-p}{p}\right)^2 \cdot \pi_2 + \frac{1-p}{p} \cdot \pi_2 + \pi_2$$

$$\begin{aligned}
\pi_2 &= \frac{1}{\left(\frac{1-p}{p}\right)^2 + \frac{1-p}{p} + 1} \\
&= \frac{1}{\frac{(1-p)^2 + p \cdot (1-p) + p^2}{p^2}} \\
&= \frac{p^2}{(1-p)^2 + p} \\
&= \frac{p^2}{p^2 - p + 1}
\end{aligned}$$

$$\begin{aligned}
\pi_1 &= \frac{1-p}{p} \cdot \pi_2 \\
&= \frac{1-p}{p} \cdot \frac{p^2}{p^2 - p + 1} \\
&= \frac{(1-p) \cdot p}{p^2 - p + 1}
\end{aligned}$$

$$\begin{aligned}
\pi_0 &= \frac{1-p}{p} \cdot \pi_1 \\
&= \frac{1-p}{p} \cdot \frac{(1-p) \cdot p}{p^2 - p + 1} \\
&= \frac{(1-p)^2}{p^2 - p + 1}
\end{aligned}$$

Hence, the steady state vector is of the form:

$$\pi = \begin{pmatrix} \frac{(1-p)^2}{p^2-p+1} & \frac{(1-p)\cdot p}{p^2-p+1} & \frac{p^2}{p^2-p+1} \end{pmatrix}$$

Problem 2

(Modeling Problem) Consider a computer system with two identical processors working in parallel. Time is slotted. The system works according to the following rules:

- *In each time slot at most one new task arrives, which happens with probability* $\alpha \in (0,1)$ *per slot. Task arrivals are independent.*
- *If one processor is available, the task is immediately started at this processor.*
- *If two processors are available, the task is immediately started on processor 1.*
- *If both processors are busy, the task is lost.*
- *A single processor ends a task in a slot with probability* β*, the tasks are independent. (Hence, the event that both processors end their tasks is given by* β^2*). Ended tasks leave the system.*
- *If a new task arrives in a slot where at least one processor ends a task, the task will be served.*

Develop a TH-DTMC model for this system. Let the state variable X_n *denote the number of busy servers during time slot* n.

- *Draw a diagram showing the possible state transitions.*
- *Find the state transition probabilities and give the state transition matrix P.*
- *Find the steady-state vector.*
- *For* $\alpha = \beta = 0.01$ *compute the steady state vector and the mean utilization* $\sum_{i=0}^{2} i \cdot \pi^{(i)}$

At first, there are four different possibilities of processor loads. In the diagram below (Figure 1), the left side corresponds to processor 1 while the right side corresponds to processor 2. *The diagram does not represent the actual state transition diagram*. In other words: it only attempts to offer an easily accessible view on the issue.

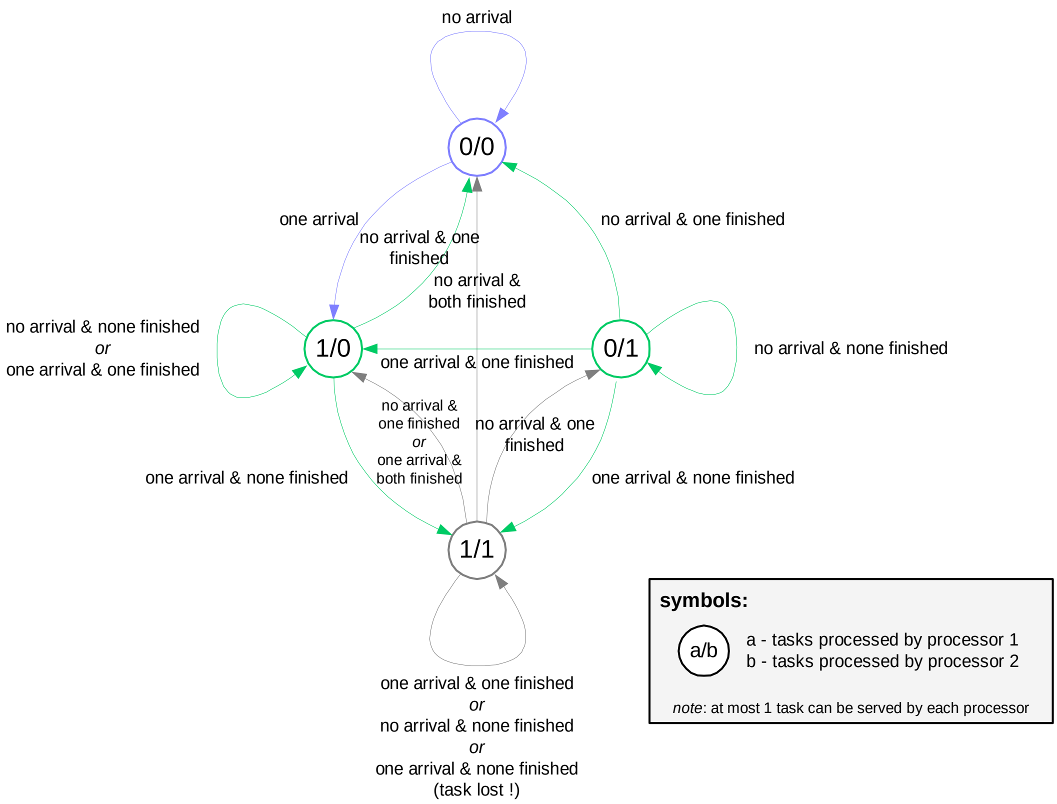


Figure 2: Overview

Indeed, we are looking for a state transition diagram where each state stands for a unique number of tasks currently processed by the system. In Figure 1 we chose the colors depending on the number of tasks: obviously, the green states 1/0 and 0/1 can be grouped reducing the complexity of the problem to just three states.

One has to be careful when finding the state transition probabilities: it is allowed that two tasks leave the system at the same time. Therefore:

$$\Pr[\text{"}\textit{no tasks arrives}\text{"}] = (1-\alpha)$$
$$\Pr[\text{"}\textit{one tasks arrives}\text{"}] = \alpha$$
$$\Pr[\text{"}\textit{no tasks finishes}\text{"}] = (1-\beta)^2$$
$$\Pr[\text{"}\textit{one tasks finishes}\text{"}] = (1-\beta)\cdot\beta$$
$$\Pr[\text{"}\textit{two tasks finish}\text{"}] = \beta^2$$

It is said that arrivals are independent and finishing tasks is as well. So it seems to be quite easy to gather all state transition probabilities in a diagram asked for:

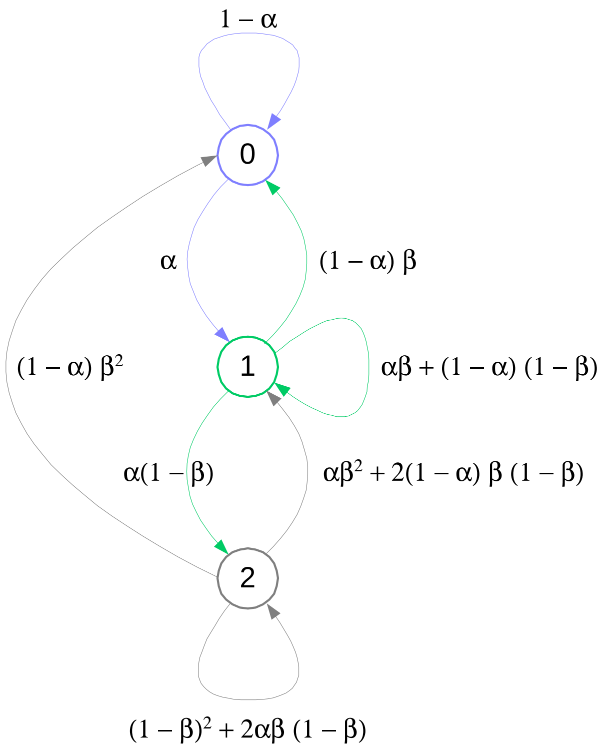


Figure 3: State transition diagram

Now that we have the state transition diagram (Figure 3) the 3x3 state transition matrix P is as follows:

$$P = \begin{pmatrix} 1-\alpha & \alpha & 0 \\ (1-\alpha)\cdot\beta & \alpha\beta + (1-\alpha)\cdot(1-\beta) & \alpha\cdot(1-\beta) \\ (1-\alpha)\cdot\beta^2 & \alpha\beta^2 + 2\cdot(1-\alpha)\cdot\beta\cdot(1-\beta) & (1-\beta)^2 + 2\alpha\beta\cdot(1-\beta) \end{pmatrix}$$

However, the steady state vector turns out to be a bit more complex than the one we found in problem 1 on page 3:

$$\pi_0 = (1-\alpha)\cdot\pi_0 + (1-\alpha)\cdot\beta\cdot\pi_1 + (1-\alpha)\cdot\beta^2\cdot\pi_2$$
$$\pi_1 = \alpha\cdot\pi_0 + (\alpha\beta + (1-\alpha)\cdot(1-\beta))\cdot\pi_1 + \left(\alpha\beta^2 + 2\cdot(1-\alpha)\cdot\beta\cdot(1-\beta)\right)\cdot\pi_2$$
$$\pi_2 = \alpha\cdot(1-\beta)\cdot\pi_1 + \left((1-\beta)^2 + 2\alpha\beta\cdot(1-\beta)\right)\cdot\pi_2$$

The last formula is solved first:

$$\begin{aligned}
\pi_2 &= \alpha\cdot(1-\beta)\cdot\pi_1 + \left((1-\beta)^2 + 2\alpha\beta\cdot(1-\beta)\right)\cdot\pi_2 \\
\left(1-(1-\beta)^2 + 2\alpha\beta\cdot(1-\beta)\right)\cdot\pi_2 &= \alpha\cdot(1-\beta)\cdot\pi_1 \\
\pi_2 &= \frac{\alpha\cdot(1-\beta)}{2\beta-\beta^2+2\alpha\beta\cdot(1-\beta)}\cdot\pi_1 \\
\pi_2 &= \frac{\alpha\cdot(1-\beta)}{\beta\cdot(2-\beta+2\alpha\cdot(1-\beta))}\cdot\pi_1
\end{aligned}$$

And the same goes for π_0:

$$\begin{aligned}
\pi_0 &= (1-\alpha)\cdot\pi_0 + (1-\alpha)\cdot\beta\cdot\pi_1 + (1-\alpha)\cdot\beta^2\cdot\pi_2 \\
\alpha\cdot\pi_0 &= (1-\alpha)\cdot\beta\cdot\pi_1 + (1-\alpha)\cdot\beta^2\cdot\pi_2 \\
\alpha\cdot\pi_0 &= (1-\alpha)\cdot\beta\cdot\pi_1 + (1-\alpha)\cdot\beta^2\cdot\frac{\alpha\cdot(1-\beta)}{\beta\cdot(2-\beta+2\alpha\cdot(1-\beta))}\cdot\pi_1 \\
\pi_0 &= \frac{1}{\alpha}\cdot\left((1-\alpha)\cdot\beta + \frac{\alpha\beta\cdot(1-\alpha)\cdot(1-\beta)}{2-\beta+2\alpha\cdot(1-\beta)}\right)\cdot\pi_1 \\
\pi_0 &= \frac{(1-\alpha)\cdot\beta}{\alpha}\cdot\left(1+\frac{\alpha\cdot(1-\beta)}{2-\beta+2\alpha\cdot(1-\beta)}\right)\cdot\pi_1
\end{aligned}$$

Applying the normalization condition:

$$\begin{aligned}
1 &= \pi_0 + \pi_1 + \pi_2 \\
1 &= \frac{(1-\alpha)\cdot\beta}{\alpha}\cdot\left(1+\frac{\alpha\cdot(1-\beta)}{2-\beta+2\alpha\cdot(1-\beta)}\right)\cdot\pi_1 + \cdot\pi_1 + \frac{\alpha\cdot(1-\beta)}{\beta\cdot(2-\beta+2\alpha\cdot(1-\beta))}\cdot\pi_1 \\
1 &= \pi_1\cdot\left(1+\frac{(1-\alpha)\cdot\beta}{\alpha}\cdot\left(1+\frac{\alpha\cdot(1-\beta)}{2-\beta+2\alpha\cdot(1-\beta)}\right)+\frac{\alpha\cdot(1-\beta)}{\beta\cdot(2-\beta+2\alpha\cdot(1-\beta))}\right)
\end{aligned}$$

$$\pi_1 = \left(1+\frac{(1-\alpha)\cdot\beta}{\alpha}+\frac{(1-\alpha)\cdot\beta\cdot(1-\beta)}{2-\beta+2\alpha\cdot(1-\beta)}+\frac{\alpha\cdot(1-\beta)}{\beta\cdot(2-\beta+2\alpha\cdot(1-\beta))}\right)^{-1}$$

Eliminating most of the brackets simplifies the formulas a lot, as seen on the next page.

However, the steady state vector takes up quite some space – we are forced to show the transpose of π otherwise we would need to switch to A3 paper size. Maybe some optimizations are left we did not discover yet:

$$\pi^T = \begin{pmatrix} -\dfrac{\beta^2 \cdot \left(2 - 3\alpha + \alpha^2 - \beta + 2\alpha\beta - \alpha^2\beta\right)}{-\alpha^2 - 2\alpha\beta + 3\alpha^2\beta - 2\beta^2 + 4\alpha\beta^2 - 3\alpha^2\beta^2 + \beta^3 - 2\alpha\beta^3 + \alpha^2\beta^3} \\ -\dfrac{\beta \cdot \left(2\alpha - 2\alpha^2\beta - \alpha\beta^2 + 2\alpha^2\beta^2\right)}{-\alpha^2 - 2\alpha\beta + 3\alpha^2\beta - 2\beta^2 + 4\alpha\beta^2 - 3\alpha^2\beta^2 + \beta^3 - 2\alpha\beta^3 + \alpha^2\beta^3} \\ \dfrac{\alpha^2 \cdot (\beta - 1)}{-\alpha^2 - 2\alpha\beta + 3\alpha^2\beta - 2\beta^2 + 4\alpha\beta^2 - 3\alpha^2\beta^2 + \beta^3 - 2\alpha\beta^3 + \alpha^2\beta^3} \end{pmatrix}$$

Under the assumption that $\alpha = \beta = 0.01$ the state transition matrix can be evaluated as:

$$P = \begin{pmatrix} 0.99 & 0.01 & 0 \\ 0.0099 & 0.9802 & 0.0099 \\ 0.000099 & 0.019603 & 0.980298 \end{pmatrix}$$

Therefore, the steady state vector is approximately:

$$\pi \approx \begin{pmatrix} 0.398394349 & 0.400406544 & 0.201199106 \end{pmatrix}$$

Of course both equations $\pi = \pi \cdot P$ and $1 = \pi_0 + \pi_1 + \pi_2$ hold true for these π and P.

Finally, the mean utilization $\sum_{i=0}^{2} i \cdot \pi^{(i)}$:

$$\begin{aligned} \sum_{i=0}^{2} i \cdot \pi^{(i)} &= 0 \cdot \pi_0 + 1 \cdot \pi_1 + 2 \cdot \pi_2 \\ &= \pi_1 + 2 \cdot \pi_2 \\ &\approx 0.400406544 + 2 \cdot 0.201199106 \\ &\approx 0.802804756 \end{aligned}$$

In the long-run average we will detect about 0.8 tasks in the system which means that the average load tends to be approximately as low as 40%.

Problem 3 (Bonus)

In another problem discussed in the tutorial we developed the following matrix of a TH-DTMC:

$$P=\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ b(0;1,p) & b(1;1,p) & 0 & 0 & \cdots & 0 \\ b(0;2,p) & b(1;2,p) & b(2;2,p) & 0 & \cdots & 0 \\ \cdots & & & & \ddots & \\ b(0;N,p) & b(1;N,p) & b(2;N,p) & b(3;N,p) & \cdots & b(N;N,p) \end{pmatrix}$$

where $p\in(0,1)$ *is a parameter,* $b(k;n,p)=\binom{n}{k}\cdot p^k\cdot(1-p)^{n-k}$ *is the distribution function of the binomial distribution and* P *is an* $(N+1)\times(N+1)$ *matrix. Use* $p=0.3$, $N=10$ *and the initial state vector* $\pi_0=(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$.

Print $\pi_k=\pi_{k-1}\cdot P=\pi_0\cdot P^k$ *for* $k\in\{1,2,5,8,10\}$. *Write a program/script using a suitable mathematics package (*`maxima/xmaxima`*,* `GNU octave`*,* `scilab`*) or in your favorite programming language.*

The trial version of Maple 8 offers a great variety of mathematical functions. Especially vector and matrix computations, as needed for that bonus problem, can be implemented with just a few lines.

After defining a function b, the distribution function of binomial distribution, the construction of the matrix P can be performed. Next, π_0 is filled with its initial values. The last lines compute π_1, π_2, π_5, π_8, π_{10}.

```
> with(LinearAlgebra): p := 0.3: N := 10:
> b := (k_, n_) -> binomial(n_,k_)*p^k_*(1.-p)^(n_-k_):
> P := Matrix(N+1, N+1, 0.):
> P[1,1] := 1.:
  for i from 2 to N+1 do
    for j from 1 to i do
      P[i,j] := b(j-1, i-1);
    od;
  od;
> pi0 := Vector[row](N+1, 0): pi0[N+1] := 1.:
  for k in [1,2,5,8,10] do
    pik := (pi0 . P^k):
    printf("%3d : ( ", k);
    for i from 1 to N do printf("%010.8f, ", pik[i]); od;
    printf("%010.8f )\n", pik[N+1]);
  od;
>
                        [11 Element Row Vector]
                        [Data Type: float[8]  ]
                 pik := [Storage: rectangular ]
                        [Order: Fortran_order ]
  1 : ( 0.02824752, 0.12106082, 0.23347444, 0.26682793, 0.20012095, 0.10291935, 0.03675691, 0.00900169, 0.00144670, 0.00013778, 0.00000590 )
                        [11 Element Row Vector]
                        [Data Type: float[8]  ]
                 pik := [Storage: rectangular ]
                        [Order: Fortran_order ]
  2 : ( 0.38941612, 0.38513682, 0.17140705, 0.04520625, 0.00782416, 0.00092858, 0.00007653, 0.00000433, 0.00000016, 0.00000000, 0.00000000 )
                        [11 Element Row Vector]
                        [Data Type: float[8]  ]
                 pik := [Storage: rectangular ]
                        [Order: Fortran_order ]
  5 : ( 0.97596401, 0.02377370, 0.00026060, 0.00000169, 0.00000001, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000 )
                        [11 Element Row Vector]
                        [Data Type: float[8]  ]
                 pik := [Storage: rectangular ]
                        [Order: Fortran_order ]
  8 : ( 0.99934409, 0.00065571, 0.00000019, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000 )
                        [11 Element Row Vector]
                        [Data Type: float[8]  ]
                 pik := [Storage: rectangular ]
                        [Order: Fortran_order ]
 10 : ( 0.99994095, 0.00005905, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000 )
>
```