Assignment 2

Joost Driessen (2674286) Rohan Zonneveld (2641655)

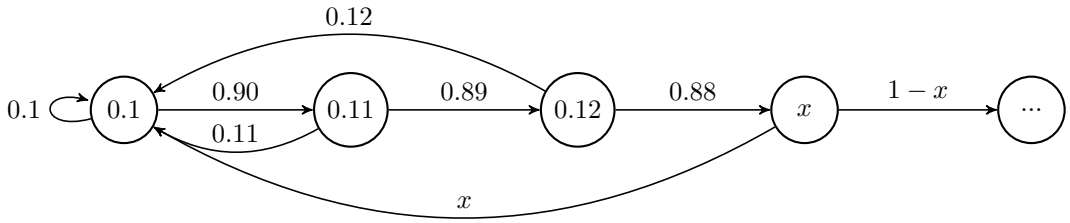
November 2023

1 Markov Decision Chain

a State space

An appropriate state space is the failure probability. This satisfies the Markov property as the future state is only dependent on the current state. When a transition to next state occurs the state (failure probability) goes up by 0.01, while in case of a failure the state returns to 0.1 and the replacement costs (reward) goes up by 1.

The MDC is depicted below.



We define $x(j)$ as the value in state j, which is equal to the failure probability. The following matrix is the result when going from one state to the next.

$$\pi_{i+1} = \left(\sum_{j=1}^{91} x(j)\pi_i(j), (1-x(1))\pi_i(1), (1-x(2))\pi_i(2), ..., (1-x(90))\pi_i(90) \right)$$

b Stationary Distribution and Long-run Average

Stationary Distribution

The definition of the stationary distribution is $\pi = P\pi$. From this formula we conclude that that after a transition the distribution does not change (it is converged so stationary). Equations 1 through 91 follow from this observation. Equation 92 results from the fact that the total probability of being in any state is always equal to one.

$$\pi(1) = \sum_{i=1}^{91} x(i)\pi(i) \tag{1}$$
$$\pi(2) = (1-x(1))\pi(1) \tag{2}$$
$$\pi(3) = (1-x(2))\pi(2) \tag{3}$$
$$\vdots$$
$$\pi(90) = (1-x(89))\pi(89) \tag{90}$$
$$\pi(91) = (1-x(90))\pi(90) \tag{91}$$
$$\sum_{i=1}^{91} \pi(i) = 1 \tag{92}$$

First we rewrite 91 to get an expression for $\pi(90)$. Then we substitute 91 into 90 and rewrite to get an expression for $\pi(89)$. The pattern continues until equation 2 in which we find an expression for $\pi(1)$:

$$\pi(90) = \frac{\pi(91)}{1 - x(90)} \tag{91}$$

$$\frac{\pi(91)}{1 - x(90)} = (1 - x(89))\pi(89) \qquad (91 \to 90)$$

$$\pi(89) = \frac{\pi(91)}{(1 - x(90))(1 - x(89))} \tag{90}$$

$$\frac{\pi(91)}{(1 - x(90))(1 - x(89))} = (1 - x(88))\pi(88) \qquad (90 \to 89)$$

$$\pi(88) = \frac{\pi(91)}{(1 - x(90))(1 - x(89))(1 - x(88))} \tag{89}$$

$$\vdots$$

$$\pi(1) = \frac{\pi(91)}{\Pi_{i=1}^{90}(1 - x(i))} \tag{2}$$

So now we have an expression for all values of π expressed in $\pi(91)$. Thus, using Equation 92, we can calculate the value $\pi(91)$ as follows:

$$\sum_{i=1}^{90} \frac{\pi(91)}{\Pi_{j=i}^{90}(1 - x(j))} = 1 \qquad (\{x \in \mathbb{N} \mid 2 \leq x \leq 91\} \to 92)$$

$$\sum_{i=1}^{90} \frac{1}{\Pi_{j=i}^{90}(1 - x(j))}\pi(91) = 1 \tag{92}$$

$$\pi(91) = \frac{1}{\sum_{i=1}^{90} \frac{1}{\Pi_{j=i}^{90}(1-x(j))}} \tag{92}$$

Using the value for $\pi(91)$ and equations 1 through 91 we can find all values of π.

$$\pi(i) = \frac{\pi(91)}{\Pi_{j=i}^{90}(1 - x(j))}$$

The process above was implemented in python to find all values of π. Additionally, a check was performed to confirm all values of π sum up to 1 (equation 92). The stationary distribution is depicted in Figure 1.

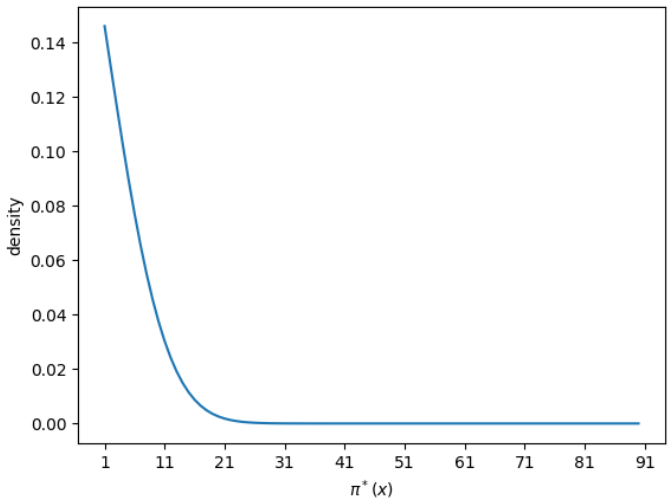


Figure 1: Density plot of the stationary distribution $\pi^*(x)$

Long-run Average Replacement Cost

The cost of a replacement is only triggered when a transition to the begin state occurs. In the long-run, the proportion of times this happens is equal to the value of $\pi(1)$, since the distribution will eventually converge. To calculate the long-run average we multiply the total probability of failure by the cost:

$$\begin{aligned}\phi^* &= \sum_x \pi(x) r(x) \\ &= \pi(1) r(1) \\ &= \pi(1) \\ &= \frac{\pi(91)}{\Pi_{j=1}^{90}(1 - x(j))} \\ &\approx 0.146\end{aligned}$$

c Simulation

The simulation of this system has been implemented in Python. First, the begin state was initialised as a vector of zeros with one as the first entry. Only the first element was set to 1, because that is the starting state, so it is certain that that is the first state of the system. The expression we found for π_{i+1} in b was implemented and applied a total of 100 times. This results in exactly the same distribution as the stationary distribution we found in b. Additionally, a check to see that all probabilities add up to one was employed.

d Poisson Equation

We begin by writing down all equations:

$$\begin{aligned} V(1) + \phi &= x(1)(1 + V(1)) + (1 - x(1))V(2) && (1) \\ V(2) + \phi &= x(2)(1 + V(1)) + (1 - x(2))V(3) && (2) \\ &\vdots \\ V(90) + \phi &= x(90)(1 + V(1)) + (1 - x(90))V(91) && (90) \\ V(91) + \phi &= x(91)(1 + V(1)) && (91) \end{aligned}$$

We have 91 equations and 92 unknowns so we can choose $V(1) = 0$. Substituting this value in Equation 91 gives and continuing the pattern gives:

$$\begin{aligned} V(91) &= 1 * (1 - 0) - \phi = 1 - \phi && (91) \\ V(90) &= 0.99 * (1 + 0) + (1 - 0.99) * (1 - \phi) - \phi = 1 - 1.01\phi && (90) \\ V(89) &= 0.98 * (1 + 0) + (1 - 0.98) * (1 - 1.01\phi) - \phi = 1 - 1.0202\phi && (89) \\ V(88) &= 0.97 * (1 + 0) + (1 - 0.97) * (1 - 1.0202\phi) - \phi = 1 - 1.0606\phi && (88) \\ &\vdots \end{aligned}$$

The coefficient (c) for ϕ follows a pattern that is defined by $c_i = (1 - x(i))c_{i+1} - 1$. Using this formula we can find a value for the coefficient of ϕ in all equations. From Equation 91 follows that the coefficient of ϕ equals -1. Iterating until Equation 1 gives the coefficient with which we can work out the correct value for ϕ.

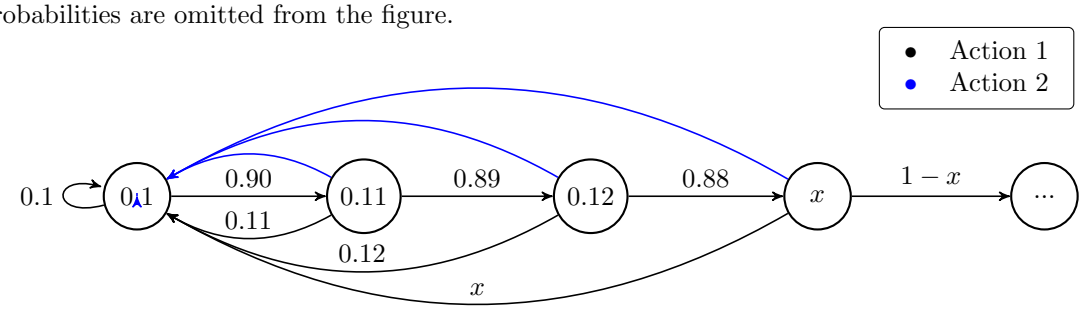
$$\begin{aligned} V(1) = 0 &= 0.1 * (1 + 0) + (1 - 0.1) * (1 - 6.494\phi) = 1 - 6.844\phi && (1) \\ \phi &= \frac{1}{6.844} && (1) \end{aligned}$$

This results in $\phi \approx 0.146$, which is the same value that was found in the simulation of the system and the Long-run Average Replacement Cost.

This rationale was implemented in python by solving the set of equations using $Ax = b$. We set V(1) to zero and define the vector x to be $(\phi, V(2), V(3), ...)$. With this vector x, A is the identity matrix with the first column replaced with the coefficients for ϕ and b equals one in every position.

e Optimal Policy

The updated MDC is depicted below. Transitions in action 2 are deterministic so the transition probabilities are omitted from the figure.



Policy Iteration

Policy Iteration consists of three steps:

1. Fix policy α
2. Solve $V + \phi = r + P_\alpha V$ for V
3. Find $\widetilde{\alpha} = \underset{\alpha'}{argmax}\{\alpha' + P_{\alpha'} V_\alpha\}$

Steps 2 and 3 are repeated until $\widetilde{\alpha} = \alpha$.

To implement this in Python the state space, the transition matrices for both actions and an initial policy of taking action one in every state were defined. Subsequently, the reward and transition matrices for the current iteration were calculated by looping over the policy and selecting the reward and transitions either from action one or action two for every state. When the policy in a state was action one the reward was defined as the value of the state. A reward (cost) is triggered only with the failure probability of that state, which is defined to be the state space. When the policy in a state is action two, the reward is 0.6.

The next step is to solve the system of equations, which is defined below:

$$V + \phi = r + PV$$
$$(I - P)V + \phi = r$$

With the definition of x, stated in d, we set $A = I - P$ with the first column of A replaced with ones, since this position does not correspond to V(1), which is zero thus can be omitted from all equations, but to ϕ. Furthermore, we set $b = r$ and solve $Ax = b$ with a solver.

Using this value matrix we calculate the value of taking both actions in every state and we select the action which results in the lowest reward, since we are trying to minimize cost.

The final step is to check if the new policy is equal to old policy. If it is then the process is stopped and the optimal policy is returned. If it is not a new iteration begins. This process is repeated until the optimal policy is found.

The optimal policy that results from this process is to choose action 1 up until state 17, after which it is more beneficial to do preventive replacement (action 2).

Value Iteration

Value iteration consists of three steps:

1. Initialise some vector V, we chose all zeros
2. Repeat $V_{t+1} = \underset{\alpha}{min}\{r_\alpha + P_\alpha V_t\}$
3. If $Span(V_{t+1} - V_t) \leq \epsilon = 1e^{-6}$ break, else continue

In every iteration we select the lowest V for every state corresponding to action one or two. This process is repeated until convergence, the optimal policy is easily retrieved by running an argmin on the latest iteration.

This algorithm was implemented in Python. Both value iteration and policy iteration resulted in the exact same optimal policy. This policy takes action 1 until state $x(17) = 0.27$ is reached, afterwards it takes action 2. This entails that from that state it becomes cheaper to replace the system prematurely, rather then taking the chances of having to replace it by chance.