Laboration 3

Johannes Pohjolainen

2021-02-16

```
set.seed(950914) #Global seed för att kunna replikera labbens resultat
```

Sammanfattning

I denna uppgift kommer vi att undersöka en födelse/döds-process som ges av ett kösystem med kapacitet 3 (en betjänas, resten i kön). Vi vill undersäla när denna kö maximalt avvisar 5% av alla förfrågningar. I Uppgift 1-4 börjar vi med att bekanta oss med processen, för att besvara frågeställningen via simulering/teoretisk i Uppgift 5/6.

Uppgift 1: Definition av vår födelse-dödsprocessen

Vi antar att ändringar i köns storlek är oberoende, och låter födelser beskrivas av en Poissonprocess med intensiteten λ per timme, medans död ses som exponentialfördet med parameter μ. Nu kan vårt system beskrivas som en kontinuerlig markovkedja med följande egenskaper:

$$\lambda_0 = \lambda_1 = \lambda_2 = \lambda, \lambda_3 = 0 \mu_1 = \mu_2 = \mu_3 = \mu, \mu_0 = 0 v_i = \lambda_i + \mu_i, (i \in [0,3]) P_{i,i+1} = \frac{\lambda_i}{v_i}, P_{i,i-1} = \frac{\mu_i}{v_i}$$

Nu när vi definierat vår process är det lämpligt att introducera de olika scenarion som kommer att testa hurvida kön är full mer än 5% av tiden (ekvivalent med regeln om avvisning):

- Scenario 1: $\lambda = 2$ och $\mu = 10$
- Scenario 1: $\lambda = 6$ och $\mu = 10$
- Scenario 1: $\lambda = 10$ och $\mu = 10$

Vi kommer alltså att undersöka hur födelseintensiteten påverkar systemets belastning vid fix dödsintensitet. Dessa scenation kommer att refereras till som S1-S3 respektive.

I S1 får vi tex (icke nämnda index har värdet 0):

$$v_0 = 2, v_3 = 10, v_i = 12, (i \in [1,3]) P_{i,i+1} = \frac{2}{v_i}, P_{i,i-1} = \frac{10}{v_i}$$

Uppgift 2: Implementation av simuleringsfunktion

För att jämföra Scenario 1-3 implementerar vi en simuleringsfunktion som gör 100 övergångar enligt givna intensiteter och sedan returnerar besökta tillstånd i kombination med besökstidpunkterna.

```
bd_process <- function(lambda, mu, initial_state = 0, steps = 100) {
  time_now <- 0
  state_now <- initial_state
  # Dessa vektorer ska byggas på i loopen nedan
  time <- 0
  state <- initial_state

  for (i in 1:steps) {
    # Systemet blir fullt med 3 i omlopp. Enligt reglerna måste vi därav förhindra kön från att växa.
    if (state_now == 3) {
      lambda_now <- 0
    } else {
      lambda_now <- lambda
    }

    # Om systemet är tomt kan vi bara fylla på
    if (state_now == 0) {
      mu_now <- 0
    } else {
      mu_now <- mu
    }

    # Detta är tiden det tar från det att vi gått in i ett stadie till att vi går till ett annat. Är i
    time_to_transition <- rexp(1, rate = mu_now+lambda_now)
    # Övergång till nästa stadie enligt övergångsmatris
    if (runif(1)<=mu_now/(mu_now+lambda_now)) {
      state_now <- state_now - 1
    } else {
      state_now <- state_now + 1
    }
    time_now <- time_now + time_to_transition # tidpunkt vid övergång
    time <- c(time, time_now) # skapar lista med tid vid alla övergångar
    state <- c(state, state_now) # skapar lista med besökta tillstånd
  }
  # Returnera en lista med de två vektorerna tid och state
  list(tid = time, state = state)
}
a = bd_process(2,1)
```

Uppgift 3: Första simulering av våra scenarion

Nu när vi har kod för att simulera enligt Scenario 1-3 kan vi testa simulera processen för våra scenarion.

```
bdplot <- function(bd) {
  plot(stepfun(bd$tid[-1], bd$state),
  do.points = FALSE,
  xlab = "Tid",
  ylab = "Tillstånd",
  main = "",
  yaxt = "n")
  axis(2, at = c(0, 1, 2, 3), las = 2)
}
```

```
s1 <- bd_process(2,10)
s2 <- bd_process(6,10)
s3 <- bd_process(10,10)
```

```
win <- par(mfrow = c(3,1))
bdplot(s1)
bdplot(s2)
bdplot(s3)
```

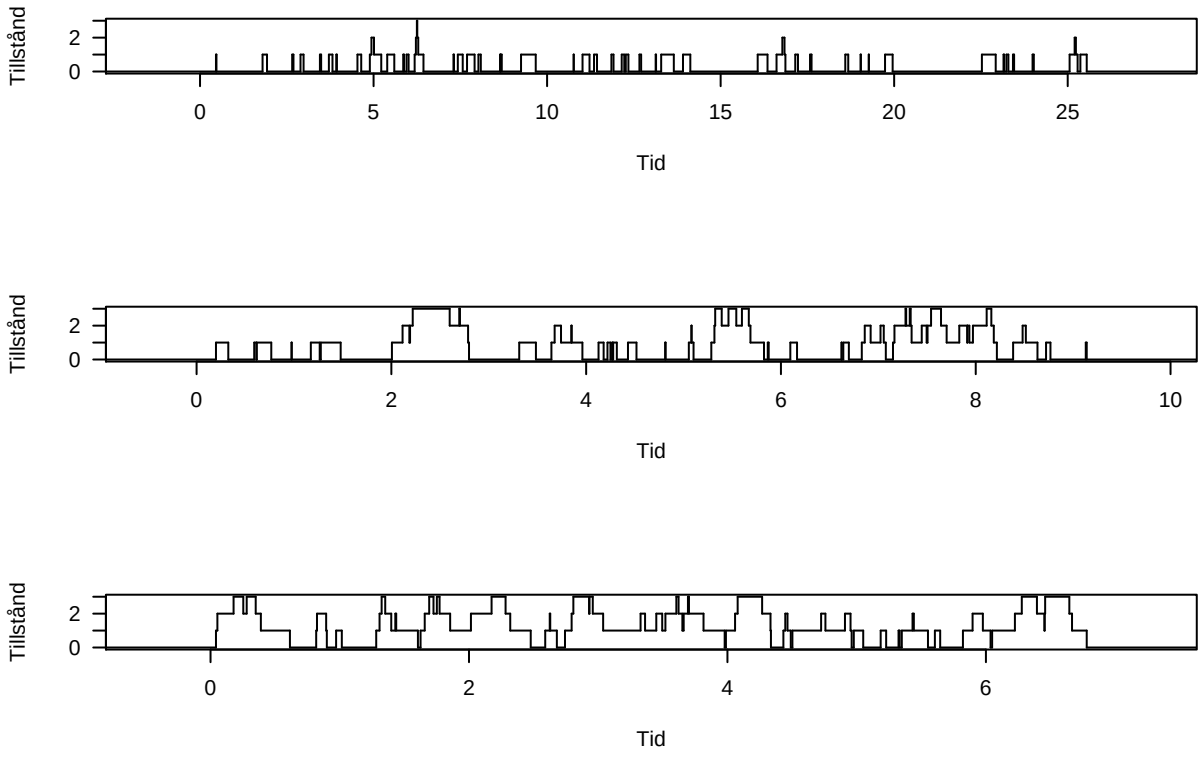

Figure 1: **Figur 1**: Visar 100 tidsteg startade med tom kö för S1-S3 respektive

Det övergripande mönstret som går att observera i Figur 1 är att det spenderas mer tid i varje tillstånd när födelseintensiteten ökar.Detta är rimligt då det leder till att fler kommer till kön och därav att kön är mindre

tom, vilket gör att de som kommer får vänta oftare.

Uppgift 4: Födelseintensitetens inverkan på tid i systemet

Om vi observerar tidsaxlarna i Figur 1 ser vi att den är kortare för högre födselsintensiteter.

För att undersöka detta vidare kommer vi att undersöka hur lång tid det tar att göra 500 tillståndsändringar i scenario 1-3.

```
ts1 <- tail(bd_process(2,10, steps = 500)$tid, 1)
ts2 <- tail(bd_process(6,10, steps = 500)$tid, 1)
ts3 <- tail(bd_process(10,10, steps = 500)$tid, 1)

Metod <- data.frame(Metod = c("S1", "S2", "S3"))
Tid <- data.frame(Tid = c(ts1, ts2, ts3))
tabell1 <- cbind(Metod, Tid)

knitr::kable(tabell1, digits = 4, caption = "Tabell 1: Visar tiden som behövdes för att göra 500 övergå
```

Table 1: Tabell 1: Visar tiden som behövdes för att göra 500 övergångar för metod S1-S3

Metod	Tid
S1	121.1108
S2	46.4316
S3	33.4078

Om vi undersöker Tabell 1 nedan kan vi se siffror som styrker att det tar mindre tid med högre födelseintensitet. Förklaringen är att en ökning innebär att vi spenderar mer tid i tillstånd där kön varken är tom eller full, och för detta tillstånd kommer det att dröja som minst innan nästa händelse sker. I dessa fall kan ju kön både avta och öka, jämfört med att man bara kan vänta på en av de händelserna. (För hög intensitet leder såklart till längre tid igen, då kön blir full mestadels av tiden istället).

Uppgift 5: Stationär fördelning via simulering

Via simulering kommer vi här undersöka hurviada endast 5% nekas.

5.1: Implementation av funktion som beräknar andel tid i tillstånd

I Uppgift 4 såg vi att mängden tid vi spenderade i processen varierar beroende på födelseintensiteten. Då kan det även vara intressant att undersöka hur mängden tid som spenderas i olika tillstånd ändras. Därav implementerar vi en funktion som beräknar andel tid spenderad i ett givet tillstånd, givet döds/födelseprocess och tillstånd.

```
proportion_in_state <- function(s, bdp) {
  tid = bdp$tid
  state = bdp$state

  stime <- 0
  for (i in 1:(length(state)-1)){
    if (state[i] == s) {
      time_stayed <- tid[i+1]-tid[i]
      stime <- stime + time_stayed
    }
  }

  stime/tail(tid,1)
}
```

5.2 Simulerad fördelning

Med kod redo är det dags att beräkna den stationära fördelningen

```
stateTimes <- function(s) {
  s1_prop = proportion_in_state(s, bd_process(2,10, step = 1000))
  s2_prop = proportion_in_state(s, bd_process(6,10, step = 1000))
  s3_prop = proportion_in_state(s, bd_process(10,10, step = 1000))

  c(s1_prop, s2_prop, s3_prop)
}

tabell2 <- data.frame(Scenario = c("S1", "S2", "S3"),
                      State0 = stateTimes(0), State1 = stateTimes(1),
                      State2 = stateTimes(2), State3 = stateTimes(3))
```

```
knitr::kable(tabell2, digits = 4, caption = "Tabell 2: Visar andelen tid som spenderades i varje tillst
```

Table 2: Tabell 2: Visar andelen tid som spenderades i varje tillstånd för alla scenarion vid 1000 steg

Scenario	State0	State1	State2	State3
S1	0.8070	0.1458	0.0283	0.0037
S2	0.4324	0.2776	0.1614	0.1040
S3	0.2725	0.2638	0.2720	0.2168

Via tabellen kan vi se att S1 verkar slå sig bäst. Enligt kravet att endast 5% av tiden fick spenderas i tillstånd 3 ser vi att endast S1 klarar sig.

Uppgift 6: Stationär fördelning via teori

För att beräkna den stationära fördelningen teoretiskt kan vi använda följande formel:

$$\pi_0 = (\sum_{i=0}^{3} p^i)^{-1} \pi_n = \pi_0 * p^n$$

Vi implementerar ovanstående formel:

```
get_stationary <- function(lambda, mu) {
  p = lambda/mu

  pi_0 <- 0
  for (i in 0:3) {
    pi_0 = pi_0 + p ^ i
  }
  pi_0 <- pi_0 ^ (-1)

  pi_0*p^(0:3)
}
```

Nu kan vi göra beräkningen från Uppgift 5 teoretiskt istället, vilket ger följande resultat:

```
stationary_df <- data.frame(rbind(get_stationary(2,10),get_stationary(6,10),get_stationary(10,10)), row
colnames(stationary_df) = paste("State", 0:3)

knitr::kable(stationary_df, digits = 4, caption = "Tabell 3: Visar andelen tid som spenderades enligt t
```

Table 3: Tabell 3: Visar andelen tid som spenderades enligt teoretiska resultat

	State 0	State 1	State 2	State 3
S1	0.8013	0.1603	0.0321	0.0064
S2	0.4596	0.2757	0.1654	0.0993
S3	0.2500	0.2500	0.2500	0.2500

De teoretiska resultaten överensstämmer med de simulerade från uppgift 5. S1 är vår vinnare.