Unreliable queue
Test of results

Ruslan Krenzler

21 Juli 2015

1 Model with maintenance

State space is $K = \{0, 1, ...N-1, b_m, b_b\}$. States $0, ..., N-1$ describe number of services after the last maintenance. After N services the system is automatically maintained. We have two blocked states $K_B = \{b_m, b_r\}$. The state b_m describes the **maintenance** state, the state b_r describes **repair** state. The failure rate in a state $k \in \{0, 1, ..., N-1\}$ is ν_k, the maintenance rate is ν_m and the repair rate is ν_r.

We assume that the input rate is λ and the queue is a $M/M/c$ queue where each server has the service rate μ_s.

```
# Some test parameters
params<-list(lambda=1,
  mu_s=1.5, #service for a single servier,
  c=3, #Number of services.
  nu=function(k){
          return(0.01*k)
  }, # Failure rates.
  nu_m=0.3, # Maintainance rate.
  nu_r=0.1, # Repair rate.
  N=5 #Number of services before mainatenance.
)
```

Let μ_s be the service rate of a single service in $M/M/c$ system then it holds

$$\mu(n) = \mu_s * \min(n, c) \tag{1}$$

```
mu<-function(params,n){
  return(params$mu_s*min(params$c,n))
}
```

For ξ - the steady state distribution of an $M/M/c$ system it holds

$$\xi(0) = \left[\sum_{j=0}^{c-1} \frac{(c*\rho)^j}{j!} + \frac{(c*\rho)^c}{c!(1-\rho)}\right]^{-1} \tag{2}$$

$$\xi(n) = \begin{cases} \frac{(c*\rho)^n}{n!}\xi(0), & \text{for } 0 < n < c \\ \frac{(\rho^n * c^c)^n}{n!}\xi(0), & \text{for } n \geq c \end{cases} \tag{3}$$

with $\rho := \frac{\lambda}{c\mu_s}$

```
xi<-function(params,n){
  if(n<0){return(0)}
  rho<-params$lambda/(params$c*params$mu_s)
  # Use formula for M/M/c from wikipedia :)
  # https://en.wikipedia.org/wiki/M/M/c_queue.
  # Keine Panik, ich habe die Formel mit Kleinrock I, Seite 404 ÃŒberprÃŒft.
  js<-(0:(params$c-1))
  xi0<-1/(sum((params$c*rho)**js/factorial(js))
          +(params$c*rho)**params$c/factorial(params$c)/(1-rho))
  if(n<params$c){
    return(xi0*(params$c*rho)**n/factorial(n))
  }else{
    return(xi0*rho**n*params$c**params$c/factorial(params$c))
  }
}
```

We define a function mapping from mathematical states K to R-language array indexes. Note, the first element of an R-array has the element index 1. We call the states r_m and r_b – "rm" and "rb" in R.

```
# The theta is stored in order
# theta(0),theta(1),...theta(N-1),theta(b_m),theta(b_m)
k_to_ri<-function(params,k){
        N<-params$N
        if(0 <= k && k <= N-1)
                return(k+1)
        if(k == "bm")
                return(N+1)
        if(k == "br")
                return(N+2)
}
```

$$\theta_N(k) := \prod_{i=1}^{k} \left(\frac{\lambda}{\nu_i + \lambda}\right)^i \theta_N(0), \qquad 0 \le k \le N-1$$

$$\theta_N(b_m) := \frac{\lambda}{\nu_m}\theta(N-1) = \frac{\lambda}{\nu_m} \prod_{i=1}^{N-1} \left(\frac{\lambda}{\nu_i + \lambda}\right)^i \theta_N(0)$$

$$\begin{aligned}\theta_N(b_r) &:= \left(\frac{(\nu_0 + \lambda)}{\nu_r} - \frac{\lambda}{\nu_r} \prod_{i=1}^{N-1} \left(\frac{\lambda}{\nu_i + \lambda}\right)^i\right) \theta_N(0) \\ &= \frac{(\nu_0 + \lambda)}{\nu_r}\theta_N(0) - \frac{\lambda}{\nu_r}\theta(N-1)\end{aligned}$$

```
theta_vector<-function(params){
  N=params$N
  retval<-rep(0,N+2)
  # set theta(0) to intermediate value 1
  # it will be normalize right before the "return" command.
  # (We also can use any non-zero value.)
  retval[k_to_ri(params,0)]<-1;
  lmbd<-params$lambda
  if(N>1){
    for(k in 1:(N-1)){
```

```
      nu<-params$nu(k)
      retval[k_to_ri(params,k)]<-retval[k_to_ri(params,k-1)]*lmbd/(lmbd+nu)
    }
  }
  retval[k_to_ri(params,"bm")]<-lmbd/params$nu_m*retval[k_to_ri(params,N-1)]
  retval[k_to_ri(params,"br")]<-(params$nu(0)+lmbd)/params$nu_r*retval[k_to_ri(params,0)]-
    params$lambda/params$nu_r*retval[k_to_ri(params,N-1)]
  # Norm the results
  retval<-retval/sum(retval)
  return(retval)
}

theta<-function(params,k){
  return(theta_vector(params)[k_to_ri(params,k)])
}
```

It holds $\pi(n,k) := \xi(n) * \theta(k)$

```
pi<-function(params,n,k){
  return(xi(params,n)*theta(params,k))
}
```

The global balance equations of the system are

$$\begin{aligned}
\pi(n,k)(1_{[n\geq 1]}\mu + \nu_k + \lambda) &= \pi(n-1,k)1_{[n\geq 1]}\lambda + \pi(n+1,k-1)1_{[k\geq 1]}\mu \\
&\quad 1_{[k=0]}\pi(n,k)(v_m + v_r), \qquad 0 \leq k \leq N-1 \\
\pi(n,b_m)\nu_m &= \pi(n+1,N-1)\mu \\
\pi(n,b_r)\nu_r &= \sum_{k\in 0,..,N-1} \pi(n,k)\nu_k
\end{aligned}$$

We check the GBEs for the proposed π. A global balance equation for particular n and k on the right hand side passes the test if the difference between right hand side (rhs) and the left hand side (lhs) is smaller then the tolerance tol. The value k used only for output.

```
check_equation<-function(lhs,rhs,k,tol){
  if(abs(lhs-rhs) < tol){
    print(sprintf("k = %s [OK]",k))
    return(TRUE)
  }else{
    print(print("k = %s [FAILED]",k))
    return(FALSE)
  }
}

test_equations<-function(params,pi,n,tol=0.000001){
  # Check equations for k=0
  # lhs <- left hand sinde
  # rhs <- right hand side.
  # ret_val < return value of the function.
  ret_val <- TRUE
  lhs <- pi(params, n,0)*(params$lambda+params$nu(0)+(n>=1)*mu(params,n))
  rhs <- pi(params, n-1,0)*(n>=1)*params$lambda+
    pi(params,n,"br")*params$nu_r+
```

```
    pi(params,n,"bm")*params$nu_m
  ret_val<-ret_val && check_equation(lhs,rhs,0,tol)


  for(k in 1:(params$N-1)){
    lhs<-pi(params,n,k)*(params$lambda+params$nu(k)+(n>=1)*mu(params,n))
    rhs<-pi(params,n-1,k)*(n>=1)*params$lambda+
      pi(params,n+1,k-1)*mu(params,n+1)
    ret_val<-ret_val && check_equation(lhs,rhs,k,tol)
  }

  lhs<-pi(params,n,"br")*params$nu_r
  rhs<-0
  for(k in 0:(params$N-1)){
    rhs<-rhs+pi(params,n,k)*params$nu(k)
  }
  ret_val<-ret_val && check_equation(lhs,rhs,"br",tol)

  lhs<-pi(params,n,"bm")*params$nu_m
  rhs<-pi(params,n+1,params$N-1)*mu(params,n+1)
  ret_val<-ret_val && check_equation(lhs,rhs,"br",tol)
  if(ret_val==TRUE){
    print(sprintf("Test for n = %d [PASSED]",n))
  }else{
    print(sprintf("Test for n = %d [FAILED]",n))
  }
  return(ret_val)
}
```

Test global balance equations

```
for(n in 0:6){
  test_equations(params, pi, n, tol=0.1)
}
```

```
## [1] "k = 0 [OK]"
## [1] "k = 1 [OK]"
## [1] "k = 2 [OK]"
## [1] "k = 3 [OK]"
## [1] "k = 4 [OK]"
## [1] "k = br [OK]"
## [1] "k = br [OK]"
## [1] "Test for n = 0 [PASSED]"
## [1] "k = 0 [OK]"
## [1] "k = 1 [OK]"
## [1] "k = 2 [OK]"
## [1] "k = 3 [OK]"
## [1] "k = 4 [OK]"
## [1] "k = br [OK]"
## [1] "k = br [OK]"
## [1] "Test for n = 1 [PASSED]"
## [1] "k = 0 [OK]"
## [1] "k = 1 [OK]"
## [1] "k = 2 [OK]"
```

```
## [1] "k = 3 [OK]"
## [1] "k = 4 [OK]"
## [1] "k = br [OK]"
## [1] "k = br [OK]"
## [1] "Test for n = 2 [PASSED]"
## [1] "k = 0 [OK]"
## [1] "k = 1 [OK]"
## [1] "k = 2 [OK]"
## [1] "k = 3 [OK]"
## [1] "k = 4 [OK]"
## [1] "k = br [OK]"
## [1] "k = br [OK]"
## [1] "Test for n = 3 [PASSED]"
## [1] "k = 0 [OK]"
## [1] "k = 1 [OK]"
## [1] "k = 2 [OK]"
## [1] "k = 3 [OK]"
## [1] "k = 4 [OK]"
## [1] "k = br [OK]"
## [1] "k = br [OK]"
## [1] "Test for n = 4 [PASSED]"
## [1] "k = 0 [OK]"
## [1] "k = 1 [OK]"
## [1] "k = 2 [OK]"
## [1] "k = 3 [OK]"
## [1] "k = 4 [OK]"
## [1] "k = br [OK]"
## [1] "k = br [OK]"
## [1] "Test for n = 5 [PASSED]"
## [1] "k = 0 [OK]"
## [1] "k = 1 [OK]"
## [1] "k = 2 [OK]"
## [1] "k = 3 [OK]"
## [1] "k = 4 [OK]"
## [1] "k = br [OK]"
## [1] "k = br [OK]"
## [1] "Test for n = 6 [PASSED]"
```