homework 2

Topics in Statistics, Jieun Shin

2021.10.14

2.32

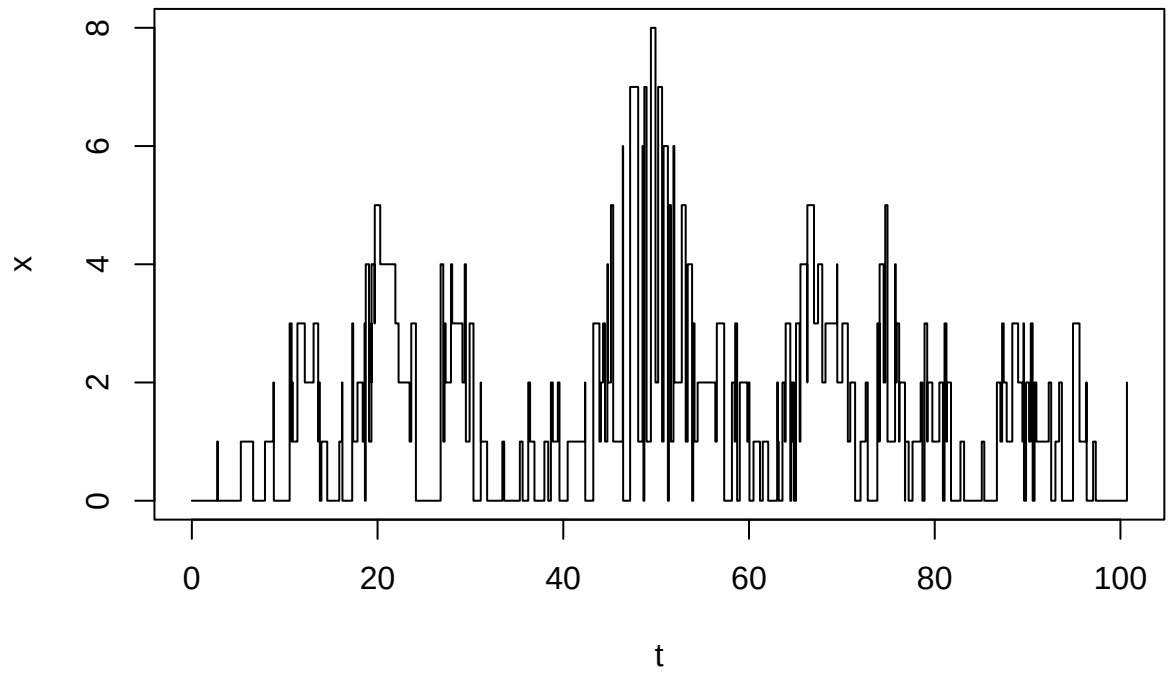
먼저 markov jump process를 생성하는 함수를 다음과 같이 정의한다.

```
# markov jump process
MarkovJumpProcess = function(n = NULL, T, birth, death){
  # N is the number of iteration
  # T is the final time
  # birth is birth rate
  # death is death rate

  # initialize
  x = t = N = y = 0
  Tn = c(0, 0)
  prob = c(death/(birth + death), birth/(birth + death))

  while( Tn[2] < T ){

    if(x[N + 1] == 0){ # 처음에는 대기 시간이 exp(birth)를 따름
      a = rexp(1, birth)
    } else{
      a = rexp(1, birth + death) # 처음 이후에는 대기 시간이 exp(birth + death)를 따름
    }

    Tn[2] = Tn[1] + a

    x = c(x, y)  # output

    if(x[N + 1] == 0){ # 처음에는 대기 시간이 exp(birth)를 따름
      y = 1
    } else{ # 처음 이후에는 대기 시간이 exp(birth + death)를 따름
      y = sample(c(x[N + 1] - 1, x[N + 1] + 1), 1, prob = prob)
    }

    N = N + 1
    Tn[1] = Tn[2]
    t = c(t, Tn[2])

    if(!is.null(n)){
      if(length(x) > n) break
    }
  }
```

```
  return(list("t" = t, "x" = x))
}
```

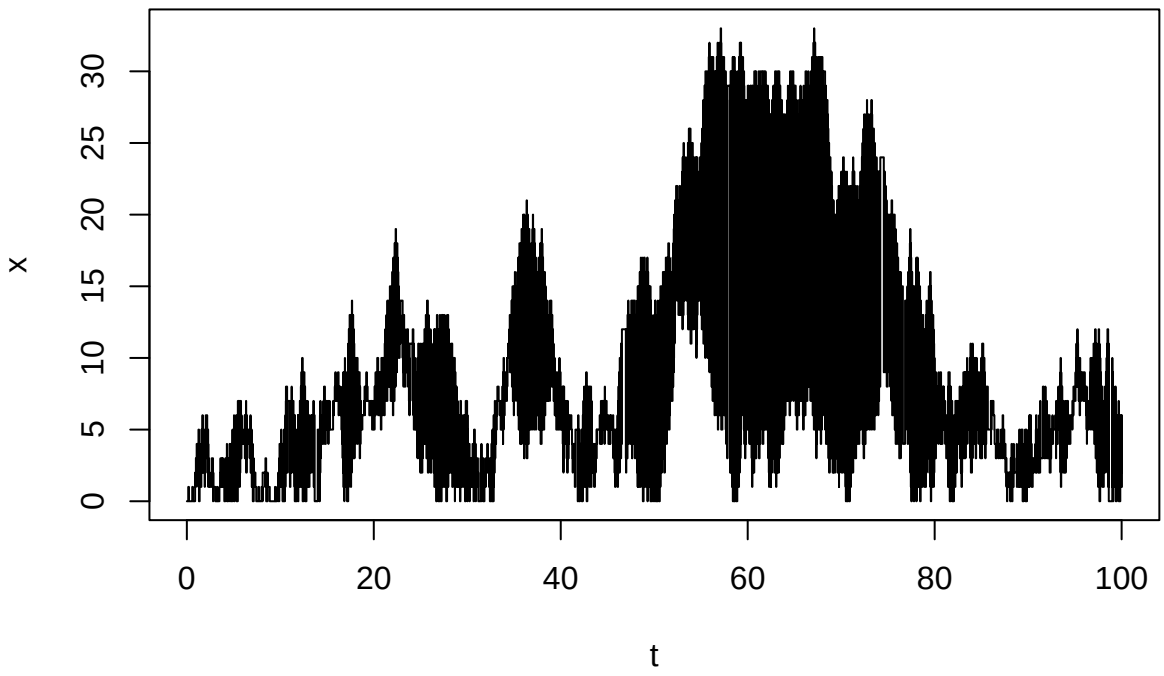
$\lambda = 1, \mu = 2$인 경우 $M/M/1$ 큐잉 모형은 다음과 같이 생성된다.

```
mjp = MarkovJumpProcess(T = 100, birth = 1, death = 2)
t = mjp$t
x = mjp$x

plot(t, x, type = 's')
```



$\lambda = 10, \mu = 11$인 경우 $M/M/1$ 큐잉 모형은 다음과 같이 생성된다.

```
mjp = MarkovJumpProcess(T = 100, birth = 10, death = 11)
t = mjp$t
x = mjp$x

plot(t, x, type = 's')
```



4.1

$l = \int_{-2}^{2} e^{-x^2/2} dx$을 추정하기 위해 다음과 같이 (A)와 (B)에 해당하는 함수들을 정의하고, Monte Carlo 시뮬레이션을 위한 함수를 정의한다.

```
Ha = function(x) 4 * exp(-x^2/2)
fa = function(n) runif(n, -2, 2)

Hb = function(x) sqrt(2 * pi) * (-2 <= x) * (x <= 2)
fb = function(n) rnorm(n)

MonteCarlo = function(N, f, H, alpha = 0.05){
  alpha = alpha

  # output
  out = list()

  x = f(N) # x generate from f(x) = U(-2, 2)
  y = H(x)

  est = mean(y)  # hat(l) = E(H(X))
  RE = sd(y) / (est * sqrt(N))  # relative error

  Z = c(qnorm(alpha / 2), qnorm(1 - alpha / 2)) * sd(y) / sqrt(N)
  ci = est + Z # confidence interval
```

```
  out$est = est
  out$RE = RE
  out$ci = ci
  return(out)
}
```

(a), (b), (c)

sample size $N = 100$ 인 경우 (A)와 (B)에서의 추정치, 상대오차, 95% 신뢰구간은 다음과 같다.

```
MonteCarlo(N = 100, f = fa, H = Ha)  # type A
```

```
## $est
## [1] 2.379219
##
## $RE
## [1] 0.04915162
##
## $ci
## [1] 2.150016 2.608422
```

```
MonteCarlo(N = 100, f = fb, H = Hb)  # type B
```

```
## $est
## [1] 2.331164
##
## $RE
## [1] 0.02757338
##
## $ci
## [1] 2.205182 2.457147
```

(d)

먼저 참값 l은 약 2.3926이다.

```
L = function(x) {exp(-x^2 / 2)}
integrate(L, -2, 2)
```

```
## 2.392576 with absolute error < 4.6e-11
```

다음으로 신뢰구간의 너비가 0.001 이하가 되도록 sample size N 값을 증가시키고 그 때 계산된 추정치를 참값과 비교해보자. 먼저 다음과 같이 시뮬레이션을 위한 함수를 지정해주었다.

```
REsim = function(N, e, f, H, alpha = 0.05){
  # N is the sample size
  # make the relative error smaller than e.
  # f and H is function corresponding to type A or type B
  out = list()
```

```
  wr = 1 # relative error
  n = 0  # iteration number
  N = N  # sample size
  err = e

  while( wr > err){
    n = n + 1
    x = f(N) # x generate from f(x) = U(-2, 2)
    y = H(x)

    wa = 2 * qnorm(1 - alpha/2) * sd(y) / sqrt(N)
    wr = wa / mean(y) # confidence interval

    # if(wr <= err) cat("n = ", n, "N = ", N)

    N = N + 100
  }

  est = mean(y)

  l = integrate(L, -2, 2)$value
  out$n = n
  out$N = N
  out$wr = wr
  out$est = est
  out$err = abs(l - est) # 참값과의 차이

  return(out)
}
```

지정한 시뮬레이션 함수를 통해 sample size N을 100씩 증가시키면서 상대오차의 너비가 0.001 이하가 되는 N과 그 반복 수, 상대오차의 너비, 추정치, 참값과의 차이를 출력한다.

```
REsim(N = 100, e = 0.01, f = fb, H = Hb, alpha = 0.05)  # type A
```

```
## $n
## [1] 71
##
## $N
## [1] 7200
##
## $wr
## [1] 0.009991087
##
## $est
## [1] 2.396125
##
## $err
## [1] 0.003548776
```

```
REsim(N = 100, e = 0.01, f = fb, H = Hb, alpha = 0.05)  # type B
```

```
## $n
## [1] 71
##
## $N
## [1] 7200
##
## $wr
## [1] 0.00961799
##
## $est
## [1] 2.403892
##
## $err
## [1] 0.01131579
```

결과를 보면 (A)유형의 경우 7000개 이상의 sample size가 필요하며, (B)유형의 경우 7000개 이하의 sample size가 필요하다. 두 방법의 추정값은 약 2.4로 비슷하지만 참값과의 차이는 (A)유형이 더 작다.

4.2

5개의 간선 $X_1, \ldots, X_5$에 대하여 만약 간선이 작동을 한다면 1, 작동을 하지 않으면 0의 값을 갖는다고 하자. 이 5개 간선의 작동 여부가 a와 b의 연결될 경우를 제공한다. 함수 $H(x)$를 a와 b가 연결된다면 1, 연결되지 않으면 0의 값을 갖는 함수라 하자.

먼저 a에서 b가 연결되는 경우를 보면 다음과 같다.

1. $x_1 = x_4 = 1$
2. $x_2 = x_5 = 1$
3. $x_1 = x_3 = x_5 = 1$
4. $x_2 = x_3 = x_4 = 1$

따라서 $x_1x_4 = 1$ or $x_2 = x_5 = 1$ or $x_1 = x_3 = x_5 = 1$ or $x_2 = x_3 = x_4 = 1$은 $H(x) = 1$일 (즉, 시스템이 잘 작동할) 필요충분조건이다.

이를 다시 표현하면 $H(x) = 1$은 $(1 - x_1x_4)(1 - x_2x_5)(1 - x_1x_3x_5)(1 - x_2x_3x_4) = 0$일 필요충분조건이다.

즉, $H(x) = 1 - (1 - x_1x_4)(1 - x_2x_5)(1 - x_1x_3x_5)(1 - x_2x_3x_4)$는 a와 b의 연결 여부를 나타내 함수가 된다.

4.4

먼저 함수 $H(x)$와 비율 p를 지정한다.

```
p = c(0.7, 0.6, 0.5, 0.4, 0.3)

# define structure function H(x)
H = function(p){
  x = c()
  for (i in 1:length(p)) x[i] = rbinom(1, 1, p[i])

  1 - (1 - x[1] * x[4]) * (1 - x[2] * x[5]) * (1 - x[1] * x[3] * x[5]) * (1 - x[2] * x[3] * x[4])
}
```

상대오차가 0.01 이하가 되는 sample size N를 찾으면 다음과 같다.

```
err = 0.01
n = 0
N = 100
x = c()
RE = 1
while( RE > err){
  n = n + 1
  for(i in 1:N) x[i] = H(p)

  est = mean(x)
  RE = sd(x) / (est * sqrt(N))

  if(RE <= err) cat("n = ", n, ", N = ", N, ", est = ",  mean(x), ", relative error = ", RE, '\n')

  N = N + 100
}
```

```
## n =  115 , N =  11500 , est =  0.4672174 , relative error =  0.009958306
```

4.5

먼저 시뮬레이션 결과를 출력할 함수를 지정한다. 함수 $H(x)$를 지정하고 결과로 점추정 값과 95% 신뢰구간을 출력한다.

먼저 (a)를 위해 $X_i, i = 1, \ldots, 5$에 대하여 감마분포의 모수 (λ_i, β_i)를 각 $(1,1), \ldots, (5,5)$로 지정한다.

마찬가지로 (b)를 위해, $X_i, i = 1, \ldots, 5$에 대하여 베르누이 분포의 모수 p_i를 각 $1/2, 1/4, 1/6, 1/8, 1/10$로 지정한다.

```
simFun = function(N, type, alpha = 0.05){
  # N is replication

  out = list()

  H = function(x) {
      min(c(x[1] + x[2], x[1] + x[4] + x[5], x[3] + x[4]))
  }

  # define the Gamma and Beta distribution
  f = function(param, type = type){

    if(type == "A"){
      param = c(1, 2, 3, 4, 5)
      x = rgamma(param, param, 5)
    }

    if(type == "B"){
      param = 1 / (2 * c(1, 2, 3, 4, 5))
      x = rbinom(5, 1, prob = param)
    }

    return(H(x))
```

```
  }



  l = replicate(N, f(param, t = "A"))

  # point estimate
  out$est = mean(l)

  # confidence interval
  out$ci = mean(l) + c(qnorm(1 - alpha/2), qnorm(alpha/2)) * sd(l) / sqrt(N)

  return(out)
}
```

(a)와 (b) 모두 점추정값이 약 0.57이며, 신뢰구간도 다음의 결과와 같이 출력된다.

```
simFun(N = 1000, type = "A", alpha = 0.05)
```

```
## $est
## [1] 0.590671
##
## $ci
## [1] 0.6106369 0.5707051
```

```
simFun(N = 1000, type = "B", alpha = 0.05)
```

```
## $est
## [1] 0.5767662
##
## $ci
## [1] 0.5955917 0.5579407
```

4.8

먼저 batch means method의 시뮬레이션을 위한 함수를 다음과 같이 지정한다.

```
Queue_batch = function(M, N, lambda, mu, err, alpha = 0.05){
  # M is run length
  # N is number of batch
  # lambda is birth rate
  # mu is death rate

  # input
  prob = c(mu/(lambda + mu), lambda/(lambda + mu))
  initM = M
  wr = 1 # relative error
  # output
  out = list()

  while(wr >= err){
```

```
    # markov jump process
    queue = MarkovJumpProcess(T = M, birth = 1, death = 2)

    x = matrix(queue$x, ncol = N) # 각 열을 배치로

    # calculate the sample mean
    y = apply(x, 2, mean)

    est = mean(y)
    ci = est + c(qnorm(1 - alpha/2), qnorm(alpha/2)) * sd(y) / sqrt(N)

    if(M == initM) CI = ci

    wa = 2 * qnorm(1 - alpha/2) * sd(y) / sqrt(N)
    wr = wa / mean(y) # confidence interval

    if(wr < err) cat("M = ", M, "relative width of CI = ", wr)

    M = M + 100
  }

  out$est = est
  out$ci = CI
  out$Mhat = M
  out$wr = wr

  return(out)
}
```

batch means method의 결과로 점추정치와 95%신뢰구간, 신뢰구간의 상대너비를 최대 0.05로 제한했을 때의 총 길이 M를 출력하면 다음과 같다.

```
Queue_batch(M = 10000, N = 30, lambda = 1, mu = 2, err = 0.05, alpha = 0.05)
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[20299]가 행의 개수[677]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[20147]가 행의 개수[672]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[20482]가 행의 개수[683]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[20807]가 행의 개수[694]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[20673]가 행의 개수[690]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[21276]가 행의 개수[710]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[20883]가 행의 개수[697]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[21606]가 행의 개수[721]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[21683]가 행의 개수[723]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[21705]가 행의 개수[724]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[21784]가 행의 개수[727]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[22327]가 행의 개수[745]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[22524]가 행의 개수[751]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[22389]가 행의 개수[747]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[22478]가 행의 개수[750]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[22762]가 행의 개수[759]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[23006]가 행의 개수[767]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[23316]가 행의 개수[778]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[23630]가 행의 개수[788]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[23599]가 행의 개수[787]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[24255]가 행의 개수[809]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[24920]가 행의 개수[831]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[24311]가 행의 개수[811]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[24785]가 행의 개수[827]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[24726]가 행의 개수[825]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[25394]가 행의 개수[847]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[25007]가 행의 개수[834]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[25407]가 행의 개수[847]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[25774]가 행의 개수[860]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[25904]가 행의 개수[864]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[26276]가 행의 개수[876]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[26918]가 행의 개수[898]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[26566]가 행의 개수[886]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[26840]가 행의 개수[895]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[27031]가 행의 개수[902]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[27019]가 행의 개수[901]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[27563]가 행의 개수[919]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[27888]가 행의 개수[930]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[27918]가 행의 개수[931]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[28005]가 행의 개수[934]의 배
## 수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[27815]가 행의 개수[928]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[28219]가 행의 개수[941]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[28546]가 행의 개수[952]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[28995]가 행의 개수[967]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[28606]가 행의 개수[954]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[29346]가 행의 개수[979]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[28954]가 행의 개수[966]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[29259]가 행의 개수[976]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[29827]가 행의 개수[995]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[29755]가 행의 개수[992]의 배
## 수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[30299]가 행의 개수[1010]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[30304]가 행의 개수[1011]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[30827]가 행의 개수[1028]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[30915]가 행의 개수[1031]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[31263]가 행의 개수[1043]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[30458]가 행의 개수[1016]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[31283]가 행의 개수[1043]의
## 배수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[31787]가 행의 개수[1060]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[31767]가 행의 개수[1059]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[31769]가 행의 개수[1059]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[32274]가 행의 개수[1076]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[32763]가 행의 개수[1093]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[32344]가 행의 개수[1079]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[33402]가 행의 개수[1114]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[33322]가 행의 개수[1111]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[33047]가 행의 개수[1102]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[33578]가 행의 개수[1120]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[33551]가 행의 개수[1119]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[34022]가 행의 개수[1135]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[33674]가 행의 개수[1123]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[34338]가 행의 개수[1145]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[34374]가 행의 개수[1146]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[34733]가 행의 개수[1158]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[35090]가 행의 개수[1170]의
## 배수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[35203]가 행의 개수[1174]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[34782]가 행의 개수[1160]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[35674]가 행의 개수[1190]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[36095]가 행의 개수[1204]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[36039]가 행의 개수[1202]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[36269]가 행의 개수[1209]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[36458]가 행의 개수[1216]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[35998]가 행의 개수[1200]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[36758]가 행의 개수[1226]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[36883]가 행의 개수[1230]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[36800]가 행의 개수[1227]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[37454]가 행의 개수[1249]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[37527]가 행의 개수[1251]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[37867]가 행의 개수[1263]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[37704]가 행의 개수[1257]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[37859]가 행의 개수[1262]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[38474]가 행의 개수[1283]의
## 배수가 되지 않습니다
```

```
## Warning in matrix(queue$x, ncol = N): 데이터의 길이[38599]가 행의 개수[1287]의
## 배수가 되지 않습니다

## Warning in matrix(queue$x, ncol = N): 데이터의 길이[38300]가 행의 개수[1277]의
## 배수가 되지 않습니다

## M =  19300 relative width of CI =  0.04952873

## $est
## [1] 1.465988
##
## $ci
## [1] 1.660779 1.462608
##
## $Mhat
## [1] 19400
##
## $wr
## [1] 0.04952873
```

이어서 regenerative method 시뮬레이션을 위한 함수를 다음과 같이 지정한다.

```
regen_ci = function(y, tau, alpha = 0.05){
    out = list()

    est = mean(y) / mean(tau)

    s11 = var(y)
    s22 = var(tau)
    s12 = cov(y, tau)

    s = sqrt( s11 - 2 * est * s12 + est^2 * s22 )
    ci = est + c(qnorm(1 - alpha/2), qnorm(alpha/2)) * s / (mean(tau) * sqrt(length(y)))

    out$s = s
    out$ci = ci

    return(out)
}

Queue_regen = function(M, lambda, mu, err, alpha = 0.05){
  # M is run length
  # lambda is birth rate
  # mu is death rate

  # input
  prob = c(mu/(lambda + mu), lambda/(lambda + mu))
  # initM = M
  # wr = 1 # relative error

  # output
  out = list()
```

```
  # while(wr >= err){
    # markov jump process
    queue = MarkovJumpProcess(T = M, birth = 1, death = 2)

    x = queue$x

    # calculate the sample mean
    reT = which(x == 0)
    y = tau = c()

    for(i in 1:length(reT)){
      a = reT[i]
      b = ifelse( i != length(reT), reT[i + 1] - 1, M)
      y[i] = sum(x[a:b])
      tau[i] = length(x[a:b])
    }

    est = mean(y) / mean(tau)

    rci = regen_ci(y, tau)
    ci = rci$ci
    s = rci$s

    # if(M == initM) CI = ci

    # wa = 2 * qnorm(1 - alpha/2) * s / sqrt(length(y))
    # wr = wa / est # confidence interval

    # if(wr < err) cat("M = ", M, "width of CI = ", wr)

    # M = M + 100
  # }

  out$est = est
  out$ci = ci
  # out$Mhat = M
  # out$wr = wr

  return(out)
}
```

regenerative method의 결과로 점추정치와 95% 신뢰구간은 다음과 같다. 신뢰구간의 상대너비를 최대 0.05로 제한했을 때의 총 길이 M를 출력하려고 했으나 시간관계상 주석처리로 결과는 생략하였다.

```
Queue_regen(M = 10000, lambda = 1, mu = 2, err = 0.05)
```

```
## $est
## [1] 1.376226
##
## $ci
## [1] 1.403754 1.348697
```