**통계적 시뮬레이션 2017311184 오준석**

**Statistical Simulation HW1**

**Results**

| **Problem 1-(1)** |
| --- |
| > test\_1 <- binom\_inverse(100, sample\_dist)  [1] 0 1 2 1 0 2 2 1 0 0 3 0 1 2 0 0 1 0 1 1 0 0 0 3 0 1 1 2 1 0 1 0 0 0 1 2 0  [38] 0 0 0 0 1 1 0 2 0 1 2 1 0 1 2 0 2 1 0 0 0 0 0 0 1 0 0 2 2 4 1 2 0 0 4 0 |

| **Problem 1-(2)** |
| --- |
| test\_2 <- binom\_dist(100, 5,0.2)  [1] 4 2 1 2 2 1 0 1 0 1 1 0 0 2 0 0 1 0 2 1 1 0 2 3 1 0 2 1 1 0 2 1 0 2 4 0 2  [38] 1 0 0 0 2 0 0 1 1 0 0 1 0 1 1 3 0 0 1 2 0 3 0 0 0 1 1 3 0 0 0 2 1 1 1 2 0 |

| **Problem 1-(3)** |
| --- |
| > summary1\_3  $`method (1) mean,var`  [1] 0.9700000 0.7162626  $`method (2) mean,var`  [1] 0.9400000 0.7842424  $`theoretical mean,var`  [1] 1.0 0.8 |

| **Problem 2** |
| --- |
| > gen\_result  [1] 5 0 4 1 0 0 2 2 3 1 0 0 2 1 2 3 0 1 3 2 0 3 3 2 0 1 2 2 5 2 1 4 3 2 3 1 2  [38] 4 1 3 1 1 3 3 4 1 1 1 7 2 1 0 1 1 1 1 0 1 4 4 4 1 3 4 2 1 3 1 0 2 2 1 2 1  > summary2  $`Theoretical mu,var`  [1] 2 2  $`Computed mu, var`  [1] 2.040000 2.018586 |

| **Problem 3-(1)** |
| --- |
| > x\_vec  [1] 0.694614804 0.834197316 0.349618154 0.370213924 0.946229012  [6] -0.379718172 0.148077594 0.483705022 0.501296656 -0.138554211  [11] 0.971196381 -0.325971607 0.566009119 -0.069317391 0.211022871  [16] 0.528844553 0.545359021 0.947360423 0.909798619 -0.807575273  [21] 0.615094050 -0.451447072 0.295926535 0.816616710 0.527366260 |

| **Problem 3-(1) E(x) and Var(X)** |
| --- |
| > summary3\_1  $mu  [1] 0.3467163  $var  [1] 0.2126065 |

| **Problem 3-(2)** |
| --- |
| > y\_vec  [1] 0.4825 0.6959 0.1222 0.1371 0.8953 0.1442 0.0219 0.2340 0.2513 0.0192  [11] 0.9432 0.1063 0.3204 0.0048 0.0445 0.2797 0.2974 0.8975 0.8277 0.6522  [21] 0.3783 0.2038 0.0876 0.6669 0.2781 0.0030 0.6901 0.0138 0.3788 0.1044  [31] 0.0013 0.2627 0.6178 0.2141 0.2760 0.1576 0.1335 0.1271 0.6738 0.6573 |

| **Problem 3-(2) E(x) and Var(X)** |
| --- |
| > summary3\_2  $mu  [1] 0.3326051  $var  [1] 0.08938587 |

| **Problem 4-(1)** |
| --- |
| > c\_val  [1] 1.5 |

| **Problem 4-(2)** |
| --- |
| > x\_vec  [1] 0.81909981 0.35705658 0.64286565 0.50142881 0.35910947 0.35587192  [7] 0.77158733 0.37585462 0.73575881 0.36741694 0.47594056 0.62252517  [13] 0.27980956 0.71369339 0.96119034 0.52079341 0.13627468 0.72463902  [19] 0.43035701 0.17958544 0.47831490 0.36133383 0.50068607 0.38181442 |

| **Problem 4-(2) E(x) and Var(X)** |
| --- |
| > summary4\_2  $mu  [1] 0.4836952  $var  [1] 0.04397117 |

| **Problem 4-(3)** |
| --- |
| > ar\_counter(100, c\_val)  [1] 1.52 |

| **Problem 5** |
| --- |
|  |

**Code**

| **Entire Code** |
| --- |
| ###---------------------------------------------------###  #Problems  ### 1. Consider the binomial distribution with a parameter(n=5, p=0.2)  ## (1) Generate 100 random numbers using the inverse transformation method  # we need distribution information.  cdf\_maker <- function(num, n, p) {  result <- rbinom(num, n, p)  cuts <- c(0, cumsum(as.vector(table(result)) / num))  return(cuts)  }  *a\_cdf* = cdf\_maker(100, 5, 0.2)  # based on the distribution info,  # we generate inverse transformed binomial random v.  binom\_inverse <- function(num, gvn\_dist) {  made\_unif <- runif(num)  temp <- cut(made\_unif, *breaks* = gvn\_dist)  tags <- (0:(nlevels(temp) - 1))  levels(temp) <- tags  return(as.vector(sapply(as.vector(temp), as.double))) # nolint # nolint  }  binom\_inverse(100,a\_cdf)  ## (2) Generate 100 random numbers using the transformation method.  # 베르누이 생성기 정의한다.  bern\_dist <- function(n, p) {  rand\_unifs <- runif(n)  turner <- function(x) {  if (x <= p) {  result <- 1  }else {  result <- 0  }  return(result)  }  return(sapply(rand\_unifs, turner))  }  # Transformation Method  binom\_dist <- function(num\_random, n, p) {  result <- c()  for (iter in 1:num\_random) {  temp <- bern\_dist(n, p)  result <- c(result, sum(temp == 1))  }  return(as.vector(result))  }  #(3) Calculate mean and variance for random numbers.  num\_rand <- 100  n <- 5  p <- 0.2  sample\_dist <- cdf\_maker(num\_rand, n, p)  test\_1 <- binom\_inverse(num\_rand, sample\_dist) ; test\_1  test\_2 <- binom\_dist(num\_rand, n, p) ; test\_2  summary1\_3 <- *list*(  'method (1) mean,var' = c(mean(test\_1), var(test\_1)),  'method (2) mean,var' = c(mean(test\_2), var(test\_2)),  'theoretical mean,var' = c(n \* p, n \* p \* (1 - p))  )  summary1\_3  ### 2. Generate 100 Poisson Numbers using inverse transformation method  inv\_pois <- function(n\_rand, lambda) {  rand\_unifs <- runif(n\_rand)  y <- 0  p <- exp(-1 \* lambda)  f\_crit <- p  poiss <- c()  for (u in rand\_unifs) {  y <- 0  p <- exp(-1 \* lambda)  f\_crit <- p  while (TRUE) {  if (u < f\_crit) {  poiss <- c(poiss, y)  break  } else {  p <- (lambda / (y + 1)) \* p  f\_crit <- f\_crit + p  y <- y + 1  }  }  }  return(poiss)  }  n\_rand <- 100  lambda <- 2  gen\_result <- inv\_pois(n\_rand, lambda)  gen\_result  summary2 <- *list*(  "Theoretical mu,var" = c(lambda, lambda),  "Computed mu, var" = c(mean(gen\_result), var(gen\_result)  ))  summary2  ### 3. Consider the pdf of the random variable X as follows.  ## (1) Generate 1,000 random numbers of X using inverse transformation method.  n\_rand <- 1000  rand\_unifs <- runif(n\_rand)  x\_vec <- c()  for (rand\_unf in rand\_unifs) {  #we gain root of the cdf of given distribution  x <- uniroot(function(x) (x^2) / 4 + x / 2 + 1 / 4 - rand\_unf,  *lower* = -1, *upper* = 1, *tol* = 0.0001)$root  x\_vec <- c(x\_vec, x)  }  x\_vec  summary3\_1 <- *list*(  "mu" = mean(x\_vec),  "var" = var(x\_vec)  )  summary3\_1  ## (2) Let Y=X^2. Estimate E(Y) and Var(Y) using the 1000 random numbrs.  # the distribution of Y will simply be distribution of X^2  y\_vec <- round(sapply(x\_vec, function(x) x^2), 4)  y\_vec  summary3\_2 <- *list*(  "mu" = mean(y\_vec),  "var" = var(y\_vec)  )  summary3\_2  ### 4. Suppose that we want to generate random numbers from gvn\_f  ## (1) Obtain min c.  gvn\_f <- *expression*(6 \* x \* (1 - x))  f\_nond <- function(x) eval(gvn\_f)  f\_d <- function(x) eval(D(gvn\_f, "x"))  max\_f <- uniroot(f\_d, *lower* = 0, *upper* = 1)  print("The minimum c is:")  c\_val <- f\_nond(max\_f$root)  c\_val  ## (2) Using the acceptance-rejection method, compute 100 rand numbers.  n\_rand <- 100 #accept-reject  #이때, c의 역수만큼. 즉, 1000개 넣으면 대략 640~660개정도가 accept.  ar\_dist <- function(n\_rand, c\_val) {  x\_vec <- c()  i <- 0  while (i < n\_rand) {  iters <- runif(1)  iters2 <- runif(1)  if (iters2 <= f\_nond(iters) / c\_val) {  x\_vec <- c(x\_vec, iters)  i <- i + 1  }  }  return(x\_vec)  }  x\_vec <- ar\_dist(n\_rand, c\_val) #result of ar method  summary4\_2 <- *list*(  "mu" = mean(x\_vec),  "var" = var(x\_vec)  )  summary4\_2  #이거랑은 별개로 근사된 distribution을 plot으로 육안 확인해보자.  x <- seq(0, 1, *by* = 0.001)  y <- f\_nond(x)  hist(x\_vec, *freq* = FALSE)  lines(x, y, *col* ='red')  ## (3) estimate average number of trials  ar\_counter <- function (n\_rand, c\_val) {  x\_vec <- c()  i <- 0  trial\_count <- 0  while (i < n\_rand) {  trial\_count <- trial\_count + 1  iters <- runif(1)  iters2 <- runif(1)  if (iters2 <= f\_nond(iters) / c\_val) {  x\_vec <- c(x\_vec, iters)  i <- i + 1  }  }  return(trial\_count / i)  }  ## The average number of trials approximate to 1.5, or c.  ar\_counter(100, c\_val)  ### 5. Generate 200 random numbers with given u prime and cov mat.  rmvn\_chol <- function(n, mu, sigma) {  #generate n random vectors from MVN(mu, sigma)  #dimension is inferred from mu and sigma  d <- length(mu) #length 2면 2차원 multivariate normal 생성.  #chol returns lower triangular. seems to be updated.  chol\_d <- chol(sigma)  Z <- matrix(rnorm(n \* d), *nrow* = n, *ncol* = d) #standard 생성.  X <- Z %\*% chol\_d + matrix(mu, *nrow* = n, *ncol* = d, *byrow* = TRUE)  return(X)  }  #choleski decomposition 사용하면, 안에서 분해 기법만 바뀐다.  #Now, use pairs to make scatterplot.  mu <- matrix(c(0, 1, 2))  cov\_mat <- matrix(c(1.0, -.5, .5, -.5, 1, -.5, .5, -.5, 1), *nrow* = length(mu))  gvn\_mults <- rmvn\_chol(200, mu, cov\_mat)  pairs(gvn\_mults) |