Assignment3functions.R

EvergreenFu

Sun Jun 26 16:59:40 2016

```r
##################################################
## dependencies
require(stats)

##################################################
## some useful functions
#  discrete uniform distribution of 0~k
rdunif <- function(n, k) floor(runif(n) * (k + 1))
#  discrete exponential distribution, bound specifies minimum value
rdexp <- function(n, rate = 0.1, bound = 0) floor(rexp(n, rate)+bound)
#  discrete normal distribution
rdnorm <- function(n, mean = 0, sd = 1) round(rnorm(n, mean, sd))

##################################################
## simulate x ~ f(x), f is continuous
#  interval:  01               x > 0 && x < 1
#             real_positive    x > 0
#             real             x is arbitrary real number
#  method:    "inverse"        FUN = F^(-1)(x)
#             "acceptance"     FUN = f
#  parameter  if known distribution is exponential, this parameter specifies lambda
#  1.instead of searching, we use optimization method to acquire c this time
#  2.if we want to use this function frequently on one single distribution, c can be preprocessed
rcont <- function(n,FUN,interval = c("01","real_positive","real"),
                  method = c("inverse","acceptance"),
                  c = NULL, parameter = NULL)
{
  if(n < 1 | !(is.function(FUN))) {
    stop("invalid input argument")
  }
  interval = match.arg(interval)
  method = match.arg(method)
  if(is.null(c))
    c = optimize(f = {if(interval == "01")FUN
                      else if(interval == "real_positive")function(x) FUN(x)/dexp(x)
                      else if(interval == "real")function(x) FUN(x)/dnorm(x)
                      },
                 maximum = T,
                 interval = {if(interval == "01")c(0,1)
                             else if(interval == "real_positive")c(0,100)
                             else if(interval == "real")c(-100,100)})$objective
  if(is.null(parameter))
    parameter = 1
  result <- if (method == "acceptance") {
    sapply(1:n,function(y){
      if (interval == "01"){
        repeat{
```

```r
          u = runif(1);v = runif(1,parameter)
          if(u < (FUN(v)/c)){return(v)}
        }
      }
      else if (interval == "real_positive"){
        repeat{
          u = runif(1);v = rexp(1)
          if(u < (FUN(v)/c/dexp(v))){return(v)}
        }
      }
      else if (interval == "real"){
        repeat{
          u = runif(1);v = rnorm(1)
          if(u < (FUN(v)/c/dnorm(v))){return(v)}
        }
      }
    })
  }
  else if(method == "inverse"){
    sapply(runif(n),FUN)
  }
  result
}

###################################################
## simulate x ~ f(x), x is discrete
#  interval:  finite              x \in [0,...,bound]
#             positive            x >= bound
#             integer             x is arbitrary integer
#  method:    "inverse"           FUN = F^(-1)(x)
#             "acceptance"        FUN = f
#  note:
#  1.if necessary we can use more accurate maximization algorithms to calculate c
#  2.if we want to use this function frequently on one single distribution, c can be preprocessed
rdisc <- function(n, FUN, interval = c("finite","positive","integer"), bound = 0,
                  method = c("inverse","acceptance"), C = NULL){
  if(n < 1 | !(is.function(FUN))) {
    stop("invalid input argument")
  }
  interval = match.arg(interval)
  method = match.arg(method)
  if(is.null(C))
    C = if (interval == "finite"){max(sapply(0:bound,FUN))*(bound+1)}
        else if (interval == "positive"){max(sapply(bound:100,FUN)/dexp(bound:100),na.rm = T)}
        else if (interval == "integer"){max(sapply(-100:100,FUN)/dnorm(-1000:100),na.rm = T)}
  result <- if (method == "acceptance") {
    sapply(1:n,function(y){
      if (interval == "finite"){
        repeat{
          u = runif(1);v = rdunif(1,(bound+1))
          if(u < (FUN(v)/C*bound)){return(v)}
        }
      }
```

```
      else if (interval == "positive"){
        repeat{
          u = runif(1);v = rdexp(1,bound = bound)
          if(u < (FUN(v)/C/dexp(v))){return(v)}
        }
      }
      else if (interval == "integer"){
        repeat{
          u = runif(1);v = rdnorm(1)
          if(u < (FUN(v)/C/dnorm(v))){return(v)}
        }
      }
    })
  }
  else if(method == "inverse"){
    floor(sapply(runif(n),FUN))
  }
  result
}
```