STATISTICS IMPLEMENTED IN R:

1. STURGE FORMULA:

STRUGE Formula is used to create a frequency table for a given data distribution. Frequency table consists of Class Intervals and the number of instances of data that fall in to corresponding intervals. So to divide the given data into equally spaced class intervals we use sturge formula.

Let x: be the given distribution of data
Find k such that 2^k > length (x). (K represent the number of intervals)
Width of each interval is (high - low)/k = width (width of each interval)
Class interval 1: low + width, Class interval 2: Class interval1 + width…

Different types of class intervals :

Exclusive Class interval	Frequency (f)
0.5-99.5	2
99.5-199.5	4
199.5-299.5	5
299.5-399.5	6
399.5-499.5	3
499.5-599.5	5
	N=25

Exclusive Range

Inclusive Class Interval	Frequency
0-99	2
100-199	4
200-299	5
300-399	6
400-499	3
500-599	5
	N = 25

Inclusive Range

STURGE IMPLEMENTATION IN R

```
# STRUGE Formula to make equal intervals for Data.

sturge <- function(vect,type = "excl")
{
  if(class(vect) == "character")
    return(transform(table(vect)))
  if(!type %in% c("incl","excl"))
    stop("Invalid type of Interval Given!")
  n <- length(vect)
  if(n == 1)return(table(vect))
  low <- round(min(vect,na.rm = TRUE))
  high <- round(max(vect,na.rm = TRUE))
  k <- round(log2(n))
  width <- round((high - low)/k)
  if(width > 0)
  {
    bins <- seq(low,high+width,width)
  }
  else
  {
    stop("Width of Interval is not Correct or NA values in
DataSet")
  }
  if(type == "incl")
  interval <- cut(vect,bins,dig.lab = 5,right = F)
  if(type == "excl")
  interval <- cut(vect,bins,dig.lab = 5)
  freq.table <- transform(table(interval))
  print(freq.table)
}
s <- sample(1:1000,50)
sturge(s,type = "incl")
```

STEP DEVIATION MEAN:

Let tab: be the given frequency table of data
Mid: Middle Values of the each class interval
A -> Assumed Mean (i.e. one of the values in mid value assumed to be mean)
d -> Deviation of each value from mean (d = $(x_i - A)/n$)
fd -> Frequency * deviation
N -> Sum of all Frequencies
n -> Total Number of observations
MEAN (x) = A + (∑fd/N)*n

STEP DEVIATION MEAN IMPLEMENTATION IN R

```
#Step Deviation Method to find the Mean.

stepdev.mean <- function(vect,type = "excl")
{
  if(!type %in% c("incl","excl"))
    stop("Invalid Type of Interval Provided")
  source("E:/Probability & Statistics/Stats Using R/functions/sturge.R")
```

```
  tab <- sturge(vect,type)
  names(tab) <- c("Interval","Frequency")
  if(type == "excl")
  interval.char <- strsplit(as.character(tab$Interval),split = "[(,]")

  if(type == "incl")
    interval.char <- strsplit(as.character(tab$Interval),split = "[[,]")

  low.limit <- as.numeric(unlist(lapply(interval.char,function(x){return(x[2])})))
  mid <- ((low.limit + width) + (low.limit))/2
  tab <- cbind(tab,MidValue = mid)
  arbitrary <- tab$MidValue[floor(length(tab$MidValue)/2)]
  dev <- (tab$MidValue - arbitrary)/nrow(tab)
  tab <- cbind(tab,Dev = round(dev,2))
  tab <- cbind(tab,FreqDev = tab$Frequency*tab$Dev)
  mean <- arbitrary + ((sum(tab$FreqDev)/sum(tab$Frequency))*nrow(tab))
  print(tab)
  cat("\nMean Using Step Deviation = ",mean)
}
```

PERMUTATION & COMIBINATION

Permutation is the calculation of all possible combinations by considering the Order. p(n,r) = n! / (n-r)!

Combination is the calculation of all possible combinations without considering the Order. c(n,r) = n! / ((n-r)! * r!)

```
# Factorial Function

fact <- function(n)
{
  if(n < 0 | !round(n) == n)
    stop("Factorial of a Negative or Decimal number is Impossible!")
  if(n == 1 | n == 0)return(1)
  else{
    options(scipen = 9999)
    return(n*fact(n-1))
  }
}

#==== Permutation
permutate <- function(n,r)
{
  if(r > n | r < 0 | n < 0)
    stop("Permutation of given arguments is Invalid !")
  else
    return(fact(n)/fact(n-r))
}

#========== Combination
combinate <- function(n,r)
{
  if(r > n | r < 0 | n < 0)
    stop("Combination of given arguments is Invalid !")
  else
    return(fact(n)/(fact(n-r)*fact(r)))
```

BINOMIAL DISTRIBUTION:

The Binomial Distribution is a discrete probability distribution where n is finite and number of outcomes is greater than 1. **$P(X = x) = c(n,x) * (p^x) * (q)^{n-x}$**

x : Event x is one of the possible outcomes of an experiment.

n : Number of trails of an experiment

p : Probability of Success of event x (q = 1-p)

For every probability distribution we have four different implementations in R.

** The value of x should be discrete i.e. x = {0,1,.....,n}. If x is continuous then probability P(X = x) is **0**

“d”	returns the height of the probability density function **(i.e. probability at a discrete point)**
“p”	returns the cumulative density function **(i.e. probability of x <= given value)**
“q”	returns the inverse cumulative density function (quantiles) **(i.e. gives x value for a probability)**
“r”	returns randomly generated numbers **(used to generate a binomial distribution)**

Note: To get a full list of the distributions available in R you can use the following command:

```
help(Distributions)        # This gives the available distribution functions in R
```

For every distribution there are four commands. The commands for each distribution are prepended with a letter to indicate the functionality. In the same line there are four functions that can be used to generate the values associated with the binomial distribution. (dbinom, pbinom, qbinom, and rbinom.)

The binomial distribution requires two extra parameters, the number of trials and the probability of success for a single trial. The commands follow the same kind of naming convention, and the names of the commands are dbinom, pbinom, qbinom, and rbinom.

DBINOM: dbinom( ) is used to give the value of probability mass function at a discrete point.

Syntax : **dbinom(x, size, prob, log = FALSE)**

x : Point at which the probability should be known, It is a vector of numbers.

size : Number of trails in an experiment.

prob : Probability of a success. (0 <= p <= 1)

Log : A Logical value if **TRUE** then **p is considered as log(p)** else p is considered as p

```
# Creating a sequence from 0 to 5
x <- seq(0,10,by = 1)
```

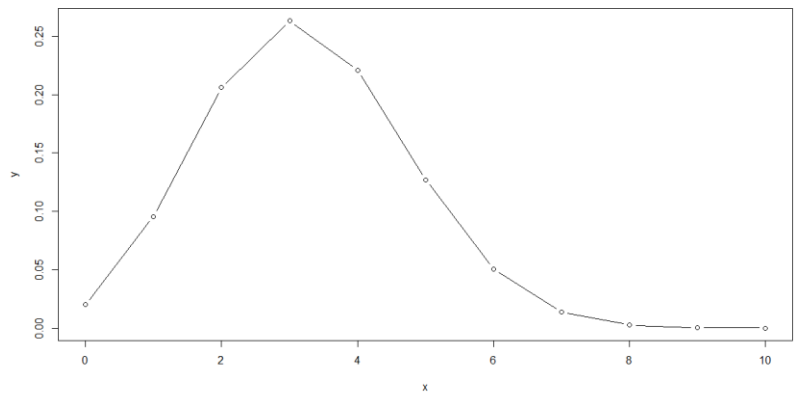
```
> # Creating a sequence from 0 to 5
> x <- seq(0,10,by = 1)
> x
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
# Finding the probability of success of each outcome with probability of success as 0.2
y <- dbinom(x,10,0.324)
```

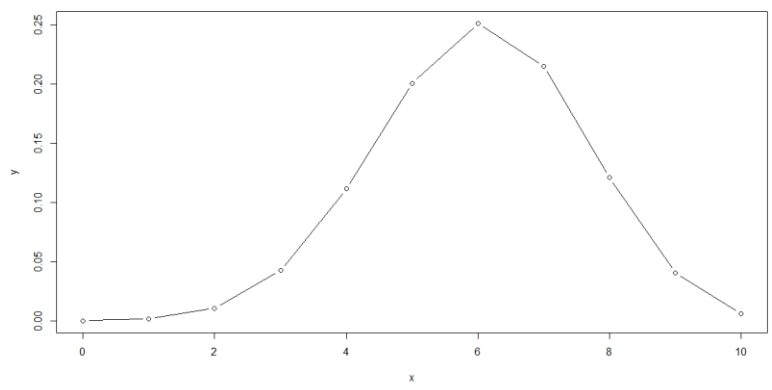
```
> # Finding the probability of success of each outcome wit
h probability of success as 0.2
> y <- dbinom(x,10,0.324)
> plot(x,y,type = "b")
> y
 [1] 0.01992814890 0.09551361305 0.20600421276
 [4] 0.26329532518 0.22084090145 0.12701618710
 [7] 0.05073131733 0.01389429148 0.00249727280
[10] 0.00026598172 0.00001274824
```

Here the probability of each discrete value in x is found using dbinom( ) function.
p(X = 0) = 0.01992814890
p(X = 1) = 0.09551361305
p(X = 8) = 0.00249727280

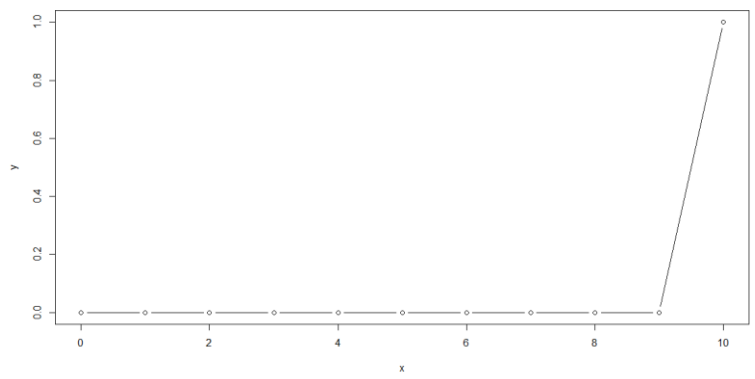
```
plot(x,y,type = "b")
```



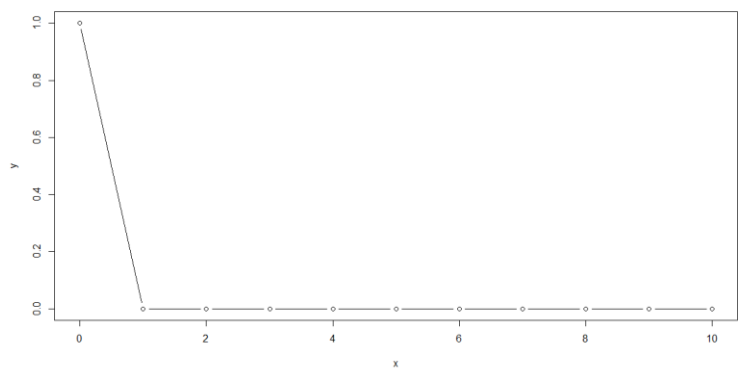

```
y <- dbinom(x,10,0.6)
plot(x,y,type = "b")
```




```
y <- dbinom(x,10,1)
plot(x,y,type = "b")
```




```
y <- dbinom(x,10,0)
plot(x,y,type = "b")
```



Next we have the cumulative probability distribution function:

```
> pbinom(24,50,0.5)
[1] 0.4438624
> pbinom(25,50,0.5)
[1] 0.5561376
> pbinom(25,51,0.5)
[1] 0.5
> pbinom(26,51,0.5)
```

[1] 0.610116

> pbinom(25,50,0.5)

[1] 0.5561376

> pbinom(25,50,0.25)

[1] 0.999962

> pbinom(25,500,0.25)

[1] 4.955658e-33

Next we have the inverse cumulative probability distribution function:

> qbinom(0.5,51,1/2)

[1] 25

> qbinom(0.25,51,1/2)

[1] 23

> pbinom(23,51,1/2)

[1] 0.2879247

> pbinom(22,51,1/2)

[1] 0.200531

Finally random numbers can be generated according to the binomial distribution:

> rbinom(5,100,.2)

[1] 30 23 21 19 18

> rbinom(5,100,.7)

[1] 66 66 58 68 63

Description

Density, distribution function, quantile function and random generation for the binomial distribution with parameters size and prob.

This is conventionally interpreted as the number of ‘successes’ in size trials.

Usage

dbinom(x, size, prob, log = FALSE)

pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)

qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)

rbinom(n, size, prob)

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
size	number of trials (zero or more).
prob	probability of success on each trial.

log, log.p　logical; if TRUE, probabilities p are given as log(p).

lower.tail　logical; if TRUE (default), probabilities are $P[X \le x]$, otherwise, $P[X > x]$.

Details

The binomial distribution with size $= n$ and prob $= p$ has density

$p(x) = choose(n, x)\ p^x\ (1-p)^{(n-x)}$

for $x = 0, \ldots, n$. Note that binomial *coefficients* can be computed by choose in R.

If an element of x is not integer, the result of dbinom is zero, with a warning.

$p(x)$ is computed using Loader's algorithm, see the reference below.

The quantile is defined as the smallest value x such that $F(x) \ge p$, where F is the distribution function.

Value

dbinom gives the density, pbinom gives the distribution function, qbinom gives the quantile function and rbinom generates random deviates.

If size is not an integer, NaN is returned.

The length of the result is determined by n for rbinom, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

Source

For dbinom a saddle-point expansion is used: see

Catherine Loader (2000). *Fast and Accurate Computation of Binomial Probabilities*; available fromhttp://www.herine.net/stat/software/dbinom.html.

pbinom uses pbeta.

qbinom uses the Cornish–Fisher Expansion to include a skewness correction to a normal approximation, followed by a search.

rbinom (for size < .Machine$integer.max) is based on

Kachitvichyanukul, V. and Schmeiser, B. W. (1988) Binomial random variate generation.*Communications of the ACM*, 31, 216–222.

For larger values it uses inversion.

See Also

Distributions for other standard distributions, including dnbinom for the negative binomial, anddpois for the Poisson distribution.

Examples

```
require(graphics)

# Compute P(45 < X < 55) for X Binomial(100,0.5)

sum(dbinom(46:54, 100, 0.5))


## Using "log = TRUE" for an extended range :

n <- 2000

k <- seq(0, n, by = 20)
```

```
plot (k, dbinom(k, n, pi/10, log = TRUE), type = "l", ylab = "log density",
      main = "dbinom(*, log=TRUE) is better than  log(dbinom(*))")
lines(k, log(dbinom(k, n, pi/10)), col = "red", lwd = 2)
## extreme points are omitted since dbinom gives 0.
mtext("dbinom(k, log=TRUE)", adj = 0)
mtext("extended range", adj = 0, line = -1, font = 4)
mtext("log(dbinom(k))", col = "red", adj = 1)
```