Duomenų analizės įvadas

2.2. dalis - R programavimas

Justas Mundeikis

VU EVAF

2019-03-24

Turinys

Loop funkcijos

Loop funkcijos

Rašant skriptus, `for`, `while` ir kiti loopai yra tinkami, bet jeigu norima parašyti kodą tiesiog konsolėje, tada susiduriama su daug problemų.

- `lapply`: loopina per list ir paleidžia funkciją kiekvienam elementui
- `sapply`: kaip ir `lapply` tik supaprastina rezultatus
- `apply`: taiko funkciją masyvo stulepliams / eilutėms
- `tapply`: taiko funkciją vektoriaus dalims
- `mapply`: multivariatinė `lapply` versija

lapply

lapply

`lapply` priima 3 argumentus: (1) list objektą, (2) funkciją arba funkcijos pavadinimą, (3) galimus funkcijos papidlomus argumentus

Jeigu `X` nėra list, tada R bando paversti `X` list objektu.

```
args(lapply)
## function (X, FUN, ...)
## NULL
lapply
## function (X, FUN, ...)
## {
##     FUN <- match.fun(FUN)
##     if (!is.vector(X) || is.object(X))
##         X <- as.list(X)
##     .Internal(lapply(X, FUN))
## }
## <bytecode: 0x564bcb7d2958>
## <environment: namespace:base>
```

lapply

`lapply` visad grąžina `list` klasės objektą

```
x <-  list(a=1:10, b=rnorm(10), c=seq(from=100, to=200, by=2))
lapply(x, mean)
## $a
## [1] 5.5
##
## $b
## [1] -0.3479987
##
## $c
## [1] 150
```

lapply

```
x <- 1:3
as.list(1:3) #taip lapply mato vektorių x konvertuus jį į list objektą
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
lapply(x, runif)
## [[1]]
## [1] 0.9501969
##
## [[2]]
## [1] 0.2764343 0.1656110
##
## [[3]]
## [1] 0.04474625 0.38590259 0.13560609
```

lapply

Išnaudojant `...` galime perleisti papildomus argumentus `runif` funckijai:

```
x <- 1:3
lapply(x, runif, min=5, max=10)
## [[1]]
## [1] 8.130686
##
## [[2]]
## [1] 7.432384 7.081280
##
## [[3]]
## [1] 9.294181 5.293083 9.988494
```

lapply

`lapply` ir kitos apply funkcijos gali naudotis anoniminėmis funkcijomis, t.y. niekur kitur nedefinuotomis funkcijomis

```
x <- list(a=matrix(1:9, nrow=3, ncol = 3),
          b=matrix(1:4, nrow = 2, ncol=2))
lapply(x, function(elt) elt[,1, drop=FALSE]) #elt yra anoniminė funkcija
## $a
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
##
## $b
##      [,1]
## [1,]    1
## [2,]    2
```

sapply

sapply

`sapply` bando supaprastinti `lapply` rezultatus (jeigu įmanoma)

- jeigu `lapply` grąžintų list, kurių kiekvienas elementas yra 1 ilgumo, tada `sapply` grąžina vektorių
- jeigu `lapply` grąžintų list, kurių kiekvienas elementas yra >1 ir vienodo ilgumo, tada `sapply` grąžina matricą
- jeigu netinka pirma du variantai, grąžina list

sapply

```
x <-  list(a=1:10, b=rnorm(10), c=seq(from=100, to=200, by=2))
lapply(x, mean)
## $a
## [1] 5.5
##
## $b
## [1] -0.5020306
##
## $c
## [1] 150
```

sapply

```
x <-  list(a=1:10, b=rnorm(10), c=seq(from=100, to=200, by=2))
sapply(x, mean)
##           a           b           c
##   5.5000000   0.1104198 150.0000000
```

apply

apply

`apply` naudojama taikyti funkcijas dataframe, matricų eilutėms ar stulpeliams. `apply` iš esmės supaprastina `for loop` naudojimą.

```
str(apply)
## function (X, MARGIN, FUN, ...)
```

apply

```
x <- matrix(1:4,2,2)
x
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
apply(x, 1, mean) #1 - eiltuėms
## [1] 2 3
apply(x, 2, mean) #2 - stulpeliams
## [1] 1.5 3.5
apply(x, 1, sum)
## [1] 4 6
apply(x, 2, sum)
## [1] 3 7
```

apply

Jeigu norima apskaičiuoti dataframe / matricų eilučių ar stuleplių sumas / vidurkius, galima naudoti jau supaprastintas funckijas, jos veikia dar greičiau, nei originalas.

- `rowSums=apply(x,1,sum)`
- `rowMeans=apply(x,1,mean)`
- `colSums=apply(x,2,sum)`
- `colMeans=apply(x,2,mean)`

apply

```
x <- matrix(rnorm(20),5,4)
apply(x, 1, quantile, probs=c(0.25 ,0.5, 0.75))
##             [,1]      [,2]      [,3]       [,4]       [,5]
## 25% -0.40986268 0.4096847 0.1138663 -0.5359259 -0.5693774
## 50% -0.07482067 0.5745440 0.3669245  0.3598979 -0.4600762
## 75%  0.20191557 0.7019729 0.5279753  1.2883942 -0.0805940
```

apply

Norint pritaikyti apply funkciją daugiau dimensijų turinčiam duomenų masyvui, būtina nurodyti vektorių, kurios dimensijos išlaikomos

```
x <- array(data=rnorm(40), dim = c(2,2,10))
apply(x, c(1,2), mean)
##             [,1]       [,2]
## [1,] -0.2592442 -0.4145625
## [2,] -0.2091574  0.1144396
```

mapply

mapply

`mapply` taiko paraleliai (vienu metu) funckiją skirtingiems argumentams

```
str(mapply)
## function (FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

- `FUN` yra funkcija, kuri bus taikoma
- `...` argumentai, kuriais naudojamasi funkcijoje
- `MoreArgs` kiti `FUN` argumentai
- `SIMPLIFY` ar rezultatas turėtų būti simplifikuotas kaip `sapply`

mapply

Jeigu norime sukurti tokį list objektą, 4 kartus rašome rep(), su argumentais 1-4 ir 4-1

```
list(rep(1,4), rep(2,3), rep(3,2), rep(4,1))
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

mapply

Supaprastinant galima naudoti `mapply` funkciją, kurios argumentai `rep` funckija ir du vektoriai 1:4 ir 4:1

```
mapply(rep, 1:4, 4:1)
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

mapply

Funkcija `noise` generuoja `n` atsitiktinių normaliojo skirstinio skaičių su vidurkiu `mean` ir standaritiniu nuokyrpiu `sd`

```
noise <- function(n, mean, sd){
        rnorm(n, mean, sd)
}

noise(4,1,2) # veikia kaip tikėtasi
## [1]  3.724385  1.729457  1.988311 -1.626522
# list(noise(1,1,0.1),noise(2,2,0.1),noise(3,3,0.1),noise(4,4,0.1))
noise(1:4,1:4,0.01) # veikia ne kaip tikėtasi
## [1] 1.016779 2.006756 2.988983 4.010547
```

mapply

Šioje vietoje galima naudotis `mapply` tam kad funkcija primtų argumentus iš vektorių

```
noise <- function(n, mean, sd){
        rnorm(n, mean, sd)
}
# list(noise(1,1,0.1),noise(2,2,0.1),noise(3,3,0.1),noise(4,4,0.1))
mapply(noise, 1:4, 1:4, 0.1)
## [[1]]
## [1] 1.049819
##
## [[2]]
## [1] 1.849216 1.977051
##
## [[3]]
## [1] 2.984417 2.921782 2.983437
##
## [[4]]
## [1] 3.972675 4.076119 4.049835 4.096907
```

tapply

tapply

Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

```
str(tapply)
## function (X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)
```

- `X` yra vektorius
- `INDEX` faktorius arba faktorių list
- `FUN` taikoma funkcija
- `...` papildomi `FUN` argumentai
- `simplify` ar supaprastinti rezultatus

tapply

```
x <- c(rnorm(10), runif(10), rnorm(10,1))
x
##  [1] -0.26845226  1.85047000  1.14226873 -0.11150875  0.55038599
##  [6]  0.09511339 -0.99012154 -0.92798749 -1.07583959  2.51063669
## [11]  0.95680986  0.07329962  0.83756126  0.62730849  0.14991190
## [16]  0.55453596  0.37646835  0.87771167  0.54843502  0.81651118
## [21]  0.14479289  1.61045287  1.05215916  2.52987214  2.40335666
## [26]  0.74914725  0.44995112 -0.78837899 -1.04261597  2.30592762
# Generate factors by specifying the pattern of their levels.
#gl(n, k, length = n*k, labels = seq_len(n), ordered = FALSE)
f <- gl(3,10)
f
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
## Levels: 1 2 3
tapply(x, f, mean)
##         1         2         3
## 0.2774965 0.5818553 0.9414665
```

tapply

```
tapply(x, f, mean, simplify = FALSE)
## $`1`
## [1] 0.2774965
##
## $`2`
## [1] 0.5818553
##
## $`3`
## [1] 0.9414665
```

tapply

```
tapply(x, f, summary)
## $`1`
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.075840 -0.763104 -0.008198  0.277496  0.994298  2.510637
##
## $`2`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0733  0.4195  0.5909  0.5819  0.8323  0.9568
##
## $`3`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.0426  0.2211  0.9007  0.9415  2.1321  2.5299
```

split

split

`split` padalina vektorių arba kitą objektą į grupes priklausomai nuo faktorių arba faktorių list

```
str(split)
## function (x, f, drop = FALSE, ...)
```

- `x` vektorius / list / dataframe
- `f` faktorius arba faktorių list
- `drop` indikuoja, ar tušti faktoriai turėtų būti panaikinti

split

```
split(x, f)
## $`1`
##  [1] -0.26845226  1.85047000  1.14226873 -0.11150875  0.55038599
##  [6]  0.09511339 -0.99012154 -0.92798749 -1.07583959  2.51063669
##
## $`2`
##  [1] 0.95680986 0.07329962 0.83756126 0.62730849 0.14991190 0.55453596
##  [7] 0.37646835 0.87771167 0.54843502 0.81651118
##
## $`3`
##  [1]  0.1447929  1.6104529  1.0521592  2.5298721  2.4033567  0.7491472
##  [7]  0.4499511 -0.7883790 -1.0426160  2.3059276
# dabar galima naudoti lapply / sapply
```

split

Taigi galime suskaidyti `x` į 3 `list` objektus ir tada kiekvienam atlikti `lapply` arba

```
lapply(split(x, f),mean)
## $`1`
## [1] 0.2774965
##
## $`2`
## [1] 0.5818553
##
## $`3`
## [1] 0.9414665
tapply(x, f, mean)
##         1         2         3
## 0.2774965 0.5818553 0.9414665
```

split

```
head(airquality)
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

split

```
s <- split(airquality, airquality$Month)
lapply(s, function(x) colMeans(x[,1:4]))
## $`5`
##    Ozone  Solar.R     Wind     Temp
##       NA       NA 11.62258 65.54839
##
## $`6`
##     Ozone   Solar.R      Wind      Temp
##        NA 190.16667  10.26667  79.10000
##
## $`7`
##      Ozone    Solar.R       Wind       Temp
##         NA 216.483871   8.941935  83.903226
##
## $`8`
##     Ozone   Solar.R      Wind      Temp
##        NA        NA  8.793548 83.967742
##
## $`9`
##    Ozone  Solar.R     Wind     Temp
```

split

```
s <- split(airquality, airquality$Month)
lapply(s, function(x) colMeans(x[,1:4], na.rm=TRUE))
## $`5`
##     Ozone   Solar.R      Wind      Temp
##  23.61538 181.29630  11.62258  65.54839
##
## $`6`
##     Ozone   Solar.R      Wind      Temp
##  29.44444 190.16667  10.26667  79.10000
##
## $`7`
##      Ozone    Solar.R       Wind       Temp
##  59.115385 216.483871   8.941935  83.903226
##
## $`8`
##      Ozone    Solar.R       Wind       Temp
##  59.961538 171.857143   8.793548  83.967742
##
## $`9`
##     Ozone   Solar.R      Wind      Temp
```

split

```
s <- split(airquality, airquality$Month)
sapply(s, function(x) colMeans(x[,1:4], na.rm=TRUE))
##                  5         6          7          8         9
## Ozone     23.61538  29.44444  59.115385  59.961538  31.44828
## Solar.R 181.29630 190.16667 216.483871 171.857143 167.43333
## Wind      11.62258  10.26667   8.941935   8.793548  10.18000
## Temp      65.54839  79.10000  83.903226  83.967742  76.90000
```

split

```
regionas <- as.factor(rep(c("Vilnius", "Kaunas", "Klaidpėda"), each=10 ))
lytis <- as.factor(c("M", "V"))
x <- data.frame(metai=rep(2013:2017),
                regionas=rep(c("Vilnius", "Kaunas", "Klaidpėda"), each=10 ),
                lytis=c("M", "V"),
                bvp=rep(runif(5,100,200),3),
                vartojimas=rnorm(30)
                )
s <- split(x, list(regionas, lytis))
```

split

```
head(x,15)
##    metai regionas lytis      bvp   vartojimas
## 1   2013  Vilnius     M 199.4630 -0.492946356
## 2   2014  Vilnius     V 157.2435  0.524036711
## 3   2015  Vilnius     M 132.9915  0.004867418
## 4   2016  Vilnius     V 112.8468  1.251074045
## 5   2017  Vilnius     M 107.8117  1.073842763
## 6   2013  Vilnius     V 199.4630 -0.488870497
## 7   2014  Vilnius     M 157.2435 -0.514454836
## 8   2015  Vilnius     V 132.9915  2.100843112
## 9   2016  Vilnius     M 112.8468 -0.089237342
## 10  2017  Vilnius     V 107.8117 -1.523286550
## 11  2013   Kaunas     M 199.4630  0.243553987
## 12  2014   Kaunas     V 157.2435  1.384000046
## 13  2015   Kaunas     M 132.9915 -0.704055916
## 14  2016   Kaunas     V 112.8468  0.141407862
## 15  2017   Kaunas     M 107.8117 -0.170962284
```

split

```
head(s,2)
## $Kaunas.M
##    metai regionas lytis      bvp vartojimas
## 11  2013   Kaunas     M 199.4630  0.2435540
## 13  2015   Kaunas     M 132.9915 -0.7040559
## 15  2017   Kaunas     M 107.8117 -0.1709623
## 17  2014   Kaunas     M 157.2435  0.6574196
## 19  2016   Kaunas     M 112.8468  1.0142053
##
## $Klaidpėda.M
##    metai  regionas lytis      bvp vartojimas
## 21  2013 Klaidpėda     M 199.4630 -0.3671821
## 23  2015 Klaidpėda     M 132.9915 -1.3453714
## 25  2017 Klaidpėda     M 107.8117  0.3748078
## 27  2014 Klaidpėda     M 157.2435  0.5830930
## 29  2016 Klaidpėda     M 112.8468 -1.7109716
```

split

```
sapply(s, function(x) mean(x[,4]))
##    Kaunas.M Klaidpėda.M   Vilnius.M    Kaunas.V Klaidpėda.V   Vilnius.V
##    142.0713    142.0713    142.0713    142.0713    142.0713    142.0713
sapply(s, function(x) colMeans(x[,4:5]))
##                Kaunas.M Klaidpėda.M     Vilnius.M    Kaunas.V Klaidpėda.V
## bvp         142.0713060 142.0713060 142.071305998 142.0713060  142.071306
## vartojimas    0.2080321  -0.4931249  -0.003585671   0.3891697    0.183771
##                Vilnius.V
## bvp         142.0713060
## vartojimas    0.3727594
```

Distribucijos

Distribucijos

Ne retai atliekant įvarius tyrimus ar skaičiuojant tikimybes statistikoje, reikės remtis tam tikrais skirstiniais. R gali generuoti įvarius skirstinius (*distributions*) `?distributions`

- dnorm
- dgamma
- beta
- dpois

ir t.t.

Distribucijos

Šioje dalyje aptarsime

- Binomial Distribution
- Poisson Distribution
- Continuous Uniform Distribution
- Exponential Distribution
- Normal Distribution

Distribucijos

Visos distribucijos galimos su 4 funkcijomis:

```
# ?dnorm

dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

- d density
- p cumulative distribution
- q quantile function
- r random number generation

set.seed(...)

Tyrimuose naudojant sugeneruotus atsitiktinius skaičius iš tam tikro skirstinio, būtina naudoti `set.seed()`, tam, kad tyrimas būtų atkartojamas.

```
set.seed(1)
rnorm(n=5, mean=5, sd=2)
## [1] 3.747092 5.367287 3.328743 8.190562 5.659016
rnorm(n=5, mean=5, sd=2)
## [1] 3.359063 5.974858 6.476649 6.151563 4.389223
set.seed(1)
rnorm(n=5, mean=5, sd=2)
## [1] 3.747092 5.367287 3.328743 8.190562 5.659016
```

Binominis skirstinys

Binominis skirstinys

Dichotomine matavimų skale matuojamų požymių reikšmių skirstinys. Skirstinys yra diskretus ir apibūdinamas parametrais n ir p. Parametras $n \geq 0$ reiškia bandymų skaičių, o p – požymio tikimybę įgyti vieną iš dviejų galimų reikšmių.

Binominio skirstinio pasiskirstymo tankio funkcija (tikimybė gauti x reikmšę su n bandymų ir p tikimybės reikmše):

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x} \text{ kur } x = 1, 2, 3, ..., n$$

Binominis skirstinys

Tarkime duomenų analizės teste yra 10 klausimų, kurių kiekvienas turi 4 galimus atsakymus, iš kurių tik vienas yra teisingas. Tarkime studentas atėjo visiškai nepasiruošęs ir visiškai atsitiktinai pasirinks atsakymus. Norint išlaikyti testą, reikia teisingai astakyti į ne mažiau kaip 5 klausimus. Kokia tikimybė, jog studentas neišlaikys testo?

- p = 1/4=0.25 ir (1-p)=1-0.25=0.75
- n = 10
- x=4

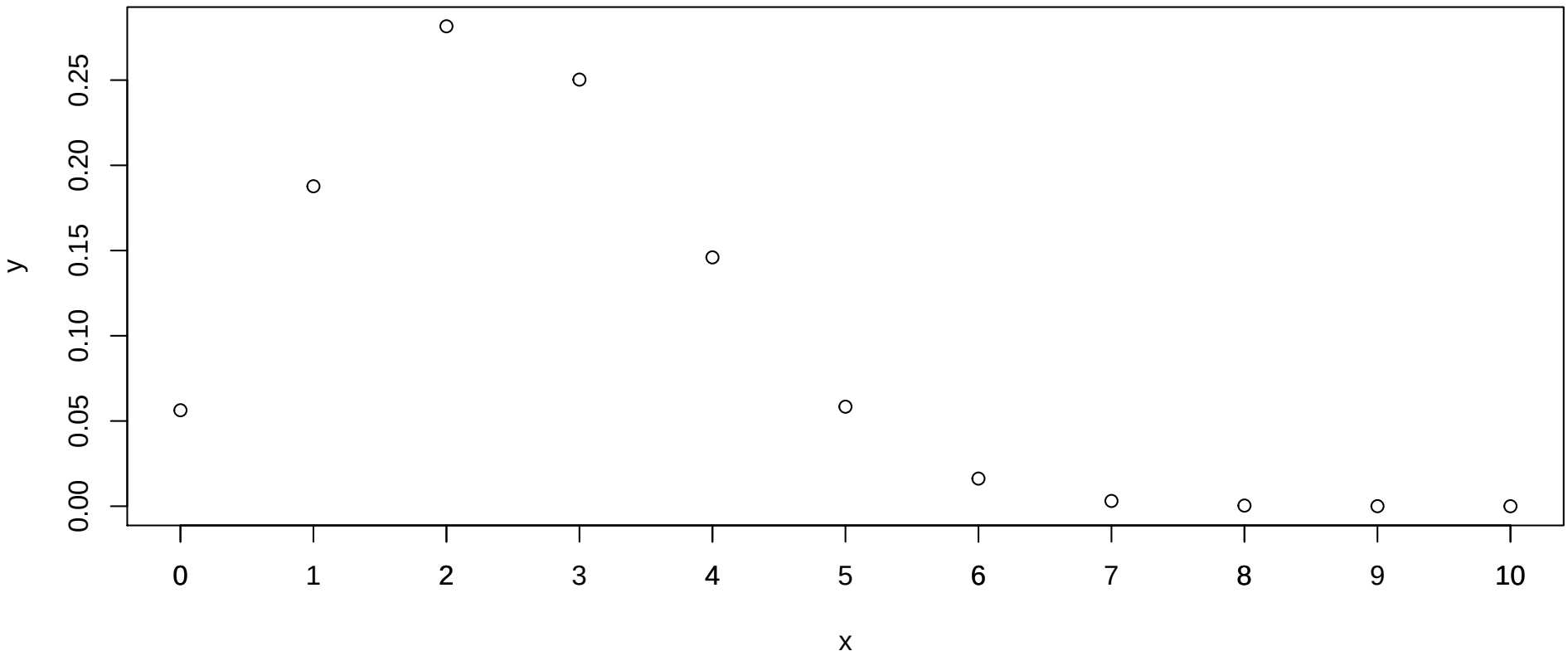
Binominis skirstinys

- p = 1/4=0.25 ir (1-p)=1-0.25=0.75
- n = 10
- x=4

```
# tikimybė jog studentas atsakys lygiai 4 teisingai
dbinom(x=4, size = 10, prob = 0.25)
## [1] 0.145998
```

Binominis skirstinys

```
x <- seq(from=0, to=10, by=1)
y <- dbinom(x, size=10, prob=0.25)
plot(x,y, type = "p")
axis(side = 1, at = x,labels = T)
```



Binominis skirstinys

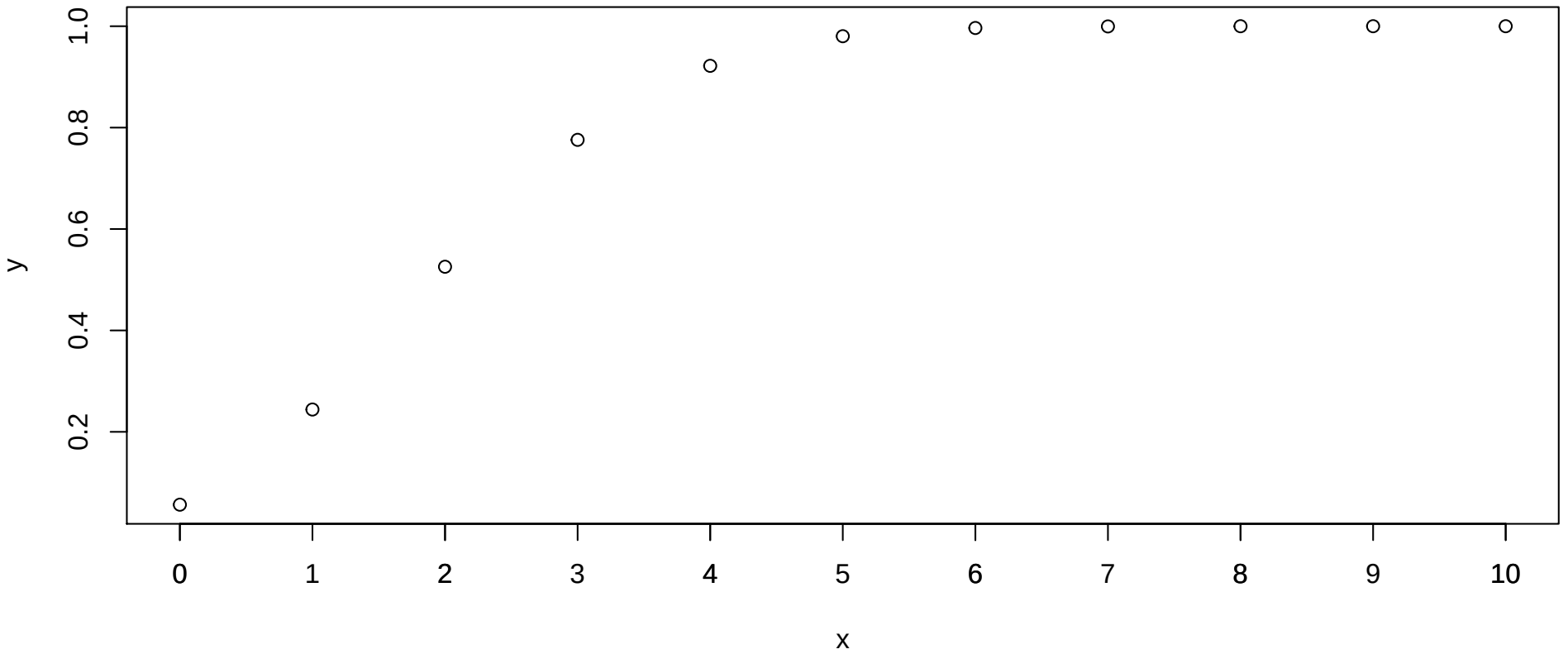
Tačiau norint žinoti visas vertes iki 4

```
# todėl norint žinoti tikimybę jog studentas atsakys į 4 arba mažiau
dbinom(x=0, size = 10, prob = 0.25)+
        dbinom(x=1, size = 10, prob = 0.25)+
        dbinom(x=2, size = 10, prob = 0.25)+
        dbinom(x=3, size = 10, prob = 0.25)+
        dbinom(x=4, size = 10, prob = 0.25)
## [1] 0.9218731

# alternatyviai galima pasinaudoti pbinom()
pbinom(q=4, size= 10, prob = 0.25, lower.tail = TRUE)
## [1] 0.9218731
# tačiau piktąjį dėstytoją domina,
# kokia tikimybė, jog studentas "praslys":
pbinom(q=4, size= 10, prob = 0.25, lower.tail = FALSE)
## [1] 0.07812691
```

Binominis skirstinys

```
x <- seq(from=0, to=10, by=1)
y <- pbinom(x, size=10, prob=0.25)
plot(x,y, type = "p")
axis(side = 1, at = x,labels = T)
```



Binominis skirstinys

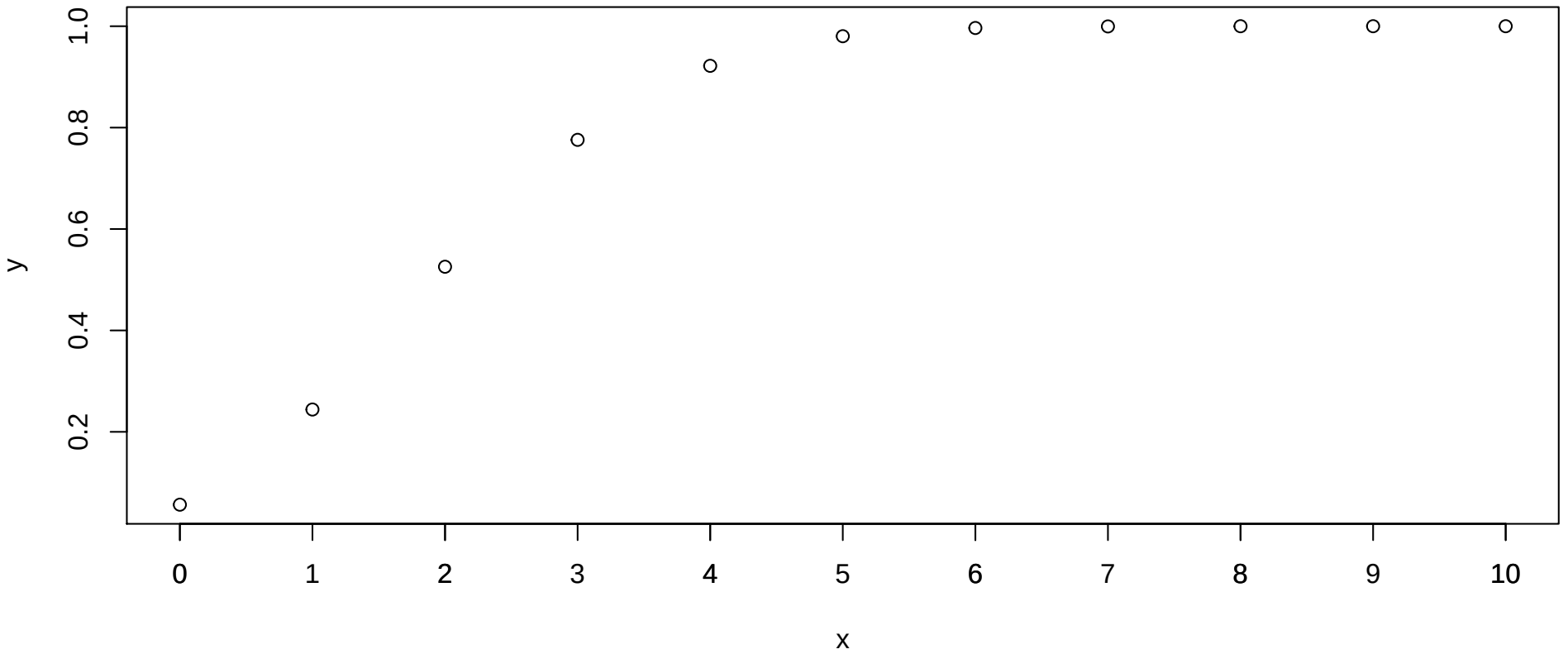
Tarkime dėstytojas nori nustatyti ribą, į kiek klausimų turi teisingai atsakyti studentai, kai:

- studentai turėdami 4 galimus pasirinkimus (daugiau alternatyvių atsakymų dėstytojas nenori sugalvoti, nes tingi)
- destyjas nenori, kad studentai praslystų pro testą didesne nei 10% tikimybe
- destytojas tingi galvoti daugiau nei 10 klausimų

```
qbinom(0.1, 10, 0.25, lower.tail = FALSE)
## [1] 4
```

Binominis skirstinys

```
x <- seq(from=0, to=10, by=1)
y <- pbinom(x, size=10, prob=0.25)
plot(x,y, type = "p")
axis(side = 1, at = x,labels = T)
```



Poisson skirstinys

Poisson skirstinys

Dichotomine matavimų skale matuojamų požymių reikšmių skirstinys. Skirstinys yra diskretus ir apibūdinamas parametrais n ir p. Parametras $n \geq 0$ reiškia bandymų skaičių, o p – požymio tikimybę įgyti vieną iš dviejų galimų reikšmių.

Poisson skirstinio pasiskirstymo tankio funkcija:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \text{ kur } x = 1, 2, 3, ..., n$$

Poisson skirstinys

Poisson distribucija

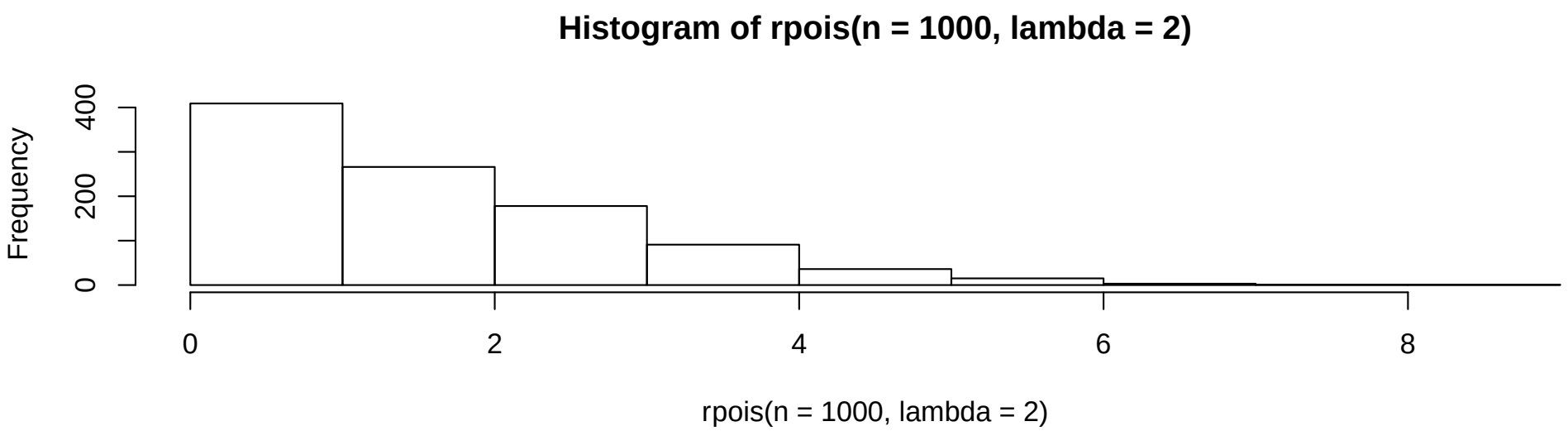
```
# ?dpois

dpois(x, lambda, log = FALSE)
ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
rpois(n, lambda)
```

Poisson skirstinys

Poisson distribucija, kur λ yra vidutinė įvykio tikimybė per tam tikrą laikotarpį

```
rpois(n=10, lambda = 1)
##  [1] 0 0 1 1 2 1 1 4 1 2
rpois(n=10, lambda=2)
##  [1] 4 1 2 0 1 1 0 1 4 1
hist(rpois(n=1000, lambda=2))
```



Poisson skirstinys

Skambučių centras per valandą sulaukia 50 skambučių. *Maximum capacity* yra 65 skambučiai per valandą. Tada skambučiai nukreipiami į alternatyvų skambučių centrą, kuriame dirba beždžionėlės, tad klientai visad lieka nepatenkinti. Klausimas, kokia yra tikimybė, jog per sekančią valandą skambučių centras sulauks: 5, 30, 60 (arba mažiau skambučių):

```
dpois(5, 50) ## 5 Pr(x=5), lambda=50
## [1] 5.022786e-16
dpois(30 , 50) ## 30  Pr(x=30), lambda=50
## [1] 0.0006771985
dpois(60, 50) ## 60  Pr(x=50), lambda=50
## [1] 0.02010487

ppois(5, 50) ## 5 arba mažiau skambučių Pr(x<=5), lambda=50
## [1] 5.567756e-16
ppois(30, 50) ## 30 arba mažiau skambučių Pr(x<=30), lambda=50
## [1] 0.001594027
ppois(60, 50) ## 60 arba mažiau skambučių Pr(x<=50),lambda=50
## [1] 0.9278398
```

Poisson skirstinys

Kokia tikimybė, jog skambučių centras sulauks daugiau skaimbučių nei skabučių centro maksimalus aptarnavimo limitas? Jeigu įmonės išsikeltas tiklsas, jog nepatenkintų klientų būtų mažiau nei 0.1%, ar patartumėte vadovybei plėsti skambučių centro galimybes? Kiek papildomų darbuotojų reikia nusamdyti skambučiui centrui, jeigu 1 darbuotojas gali priimti po 5 skambučius per valandą?

```
ppois(q=65, lambda = 50, lower.tail = TRUE)
## [1] 0.9827354
ppois(q=65, lambda = 50, lower.tail = FALSE)
## [1] 0.01726457

# Kiek papildomų skambučių reiktų papildomai galėti priimti?
qpois(p=0.001, lambda = 50, lower.tail = FALSE)
## [1] 73
ceiling(
        (qpois(p=0.001, lambda = 50, lower.tail = FALSE) - 65) / 5
        )
## [1] 2
```

Tolygusis skirstinys (Continuous uniform distribution)

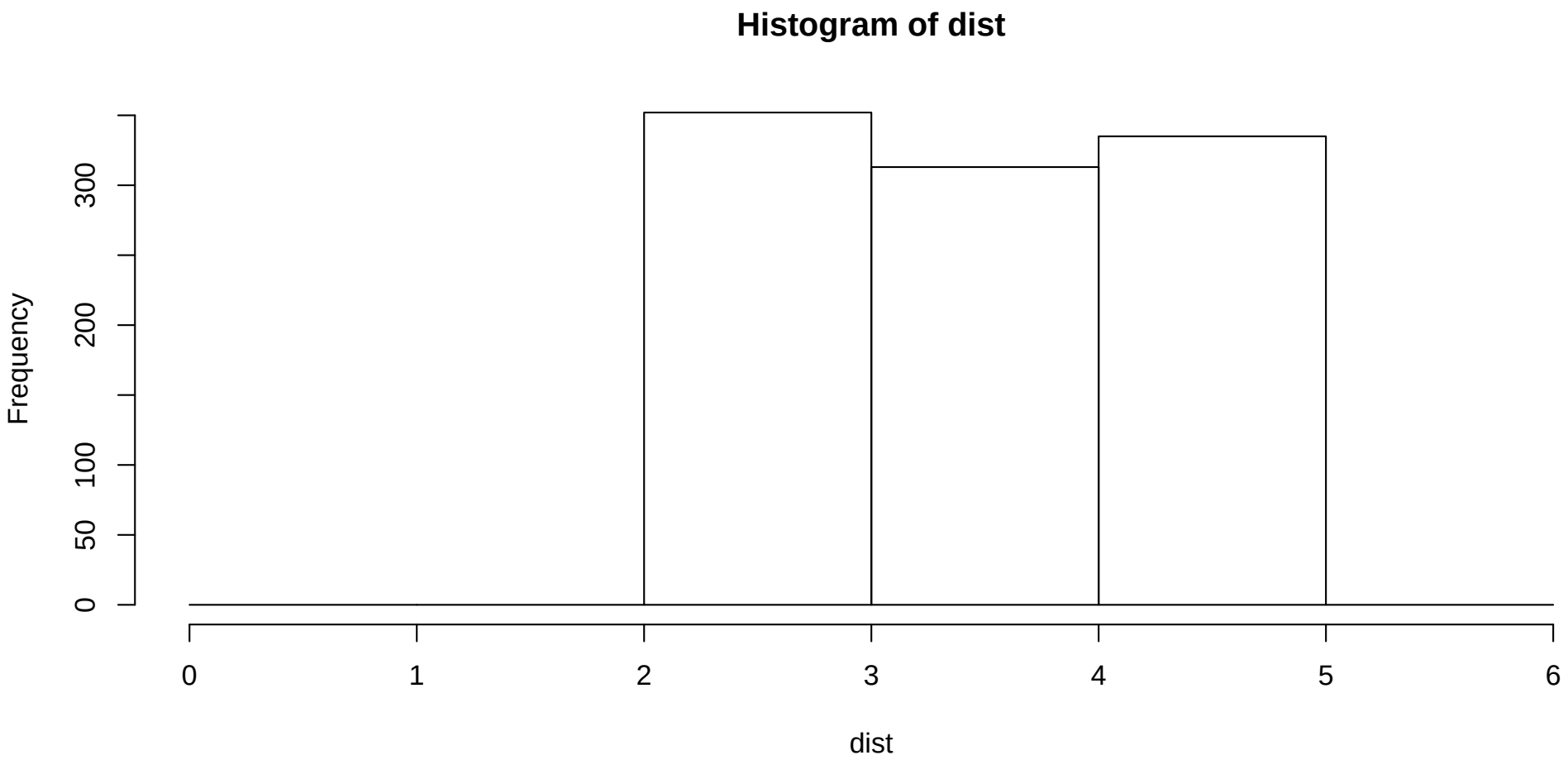
Tolygusis skirstinys (Continuous uniform distribution)

Skirstinys su vienoda tikimybe visiems skaičiams tarp a ir b. Visais kitais atvejais tikimybė =0.

$$f(x) = \begin{cases} \frac{1}{b-a} \text{ when } a \leq x \leq b \\ 0, else \end{cases}$$

Tolygusis skirstinys (Continuous uniform distribution)

```
dist <- runif(n=1000, min=2, max=5)
hist(dist, breaks = seq(from=0, to=6, by=1))
```



Eksponentinis skirstinys

Eksponentinis skirstinys

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x}, \text{ when } x \geq 0 \\ 0, x < 0 \end{cases}$$

$$f(x, \mu) = \begin{cases} \frac{1}{\mu} e^{-x/\mu}, \text{ when } x \geq 0 \\ 0, x < 0 \end{cases}$$

Eksponentinis skirstinys

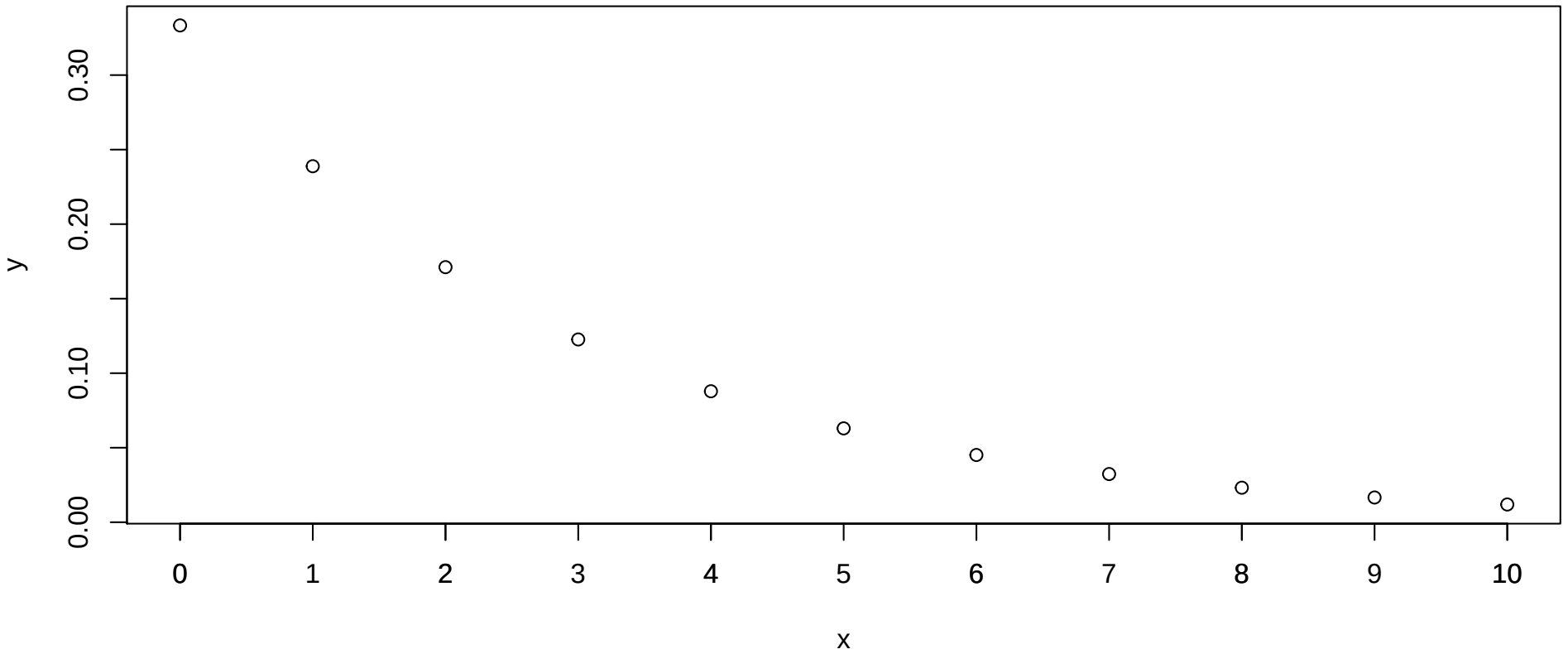
PVZ: Tarkime kasininkas aptarnauja vieną klientą per vidutiniškai 3 minutes. Žinoma, kad aptarnavimo laikas turi eksponentinį skirstinį. Kokia tikimybe sekantis klientas bus aptarnautas per mažiau nei 2 minutes

- vidutinis aptarnavimo greitis: 1/3=0.333 klientų per minutę

```
pexp(2, rate=1/3)
## [1] 0.4865829
```

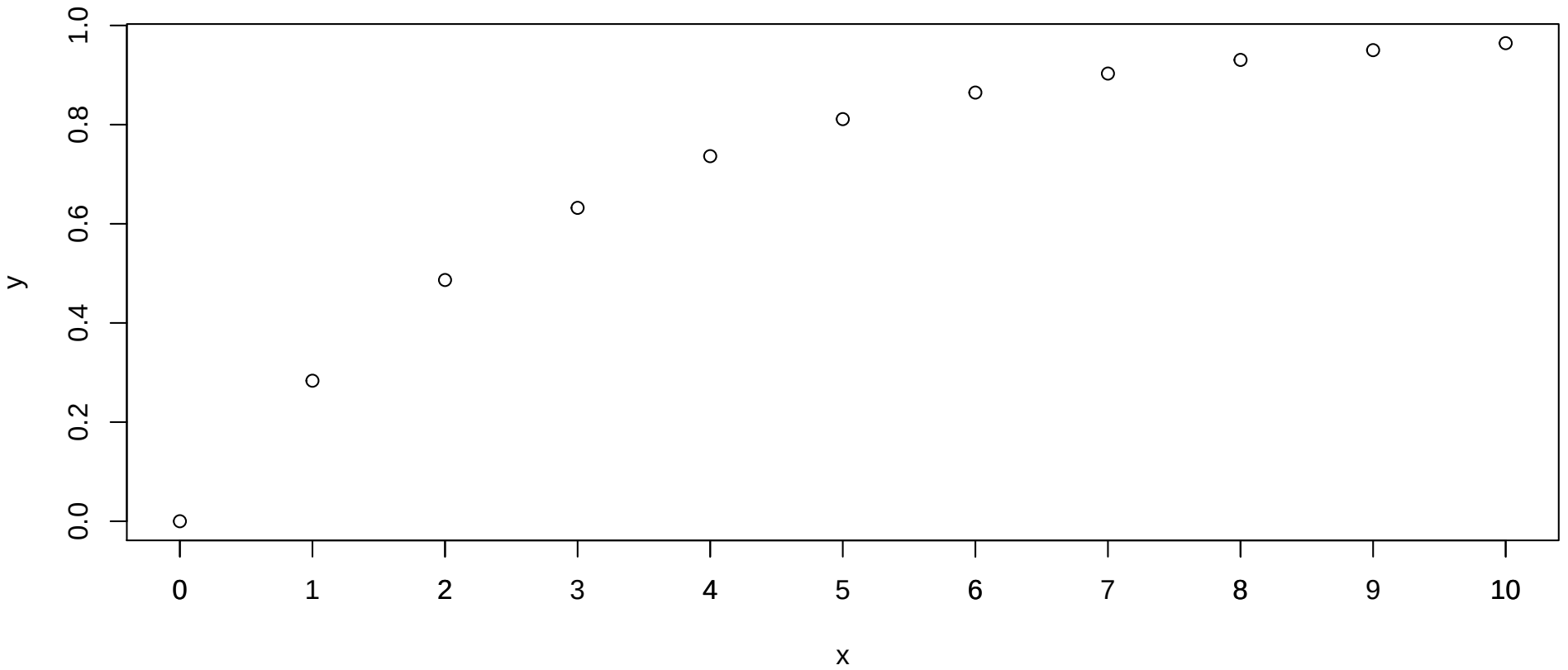
Eksponentinis skirstinys

```
x <- seq(from=0, to=10, by=1)
y <- dexp(x, rate=1/3 )
plot(x,y, type = "p")
axis(side = 1, at = x,labels = T)
```



Eksponentinis skirstinys

```
x <- seq(from=0, to=10, by=1)
y <- pexp(x, rate=1/3 )
plot(x,y, type = "p")
axis(side = 1, at = x,labels = T)
```



Normalusis skirstinys

Normalusis skirstinys

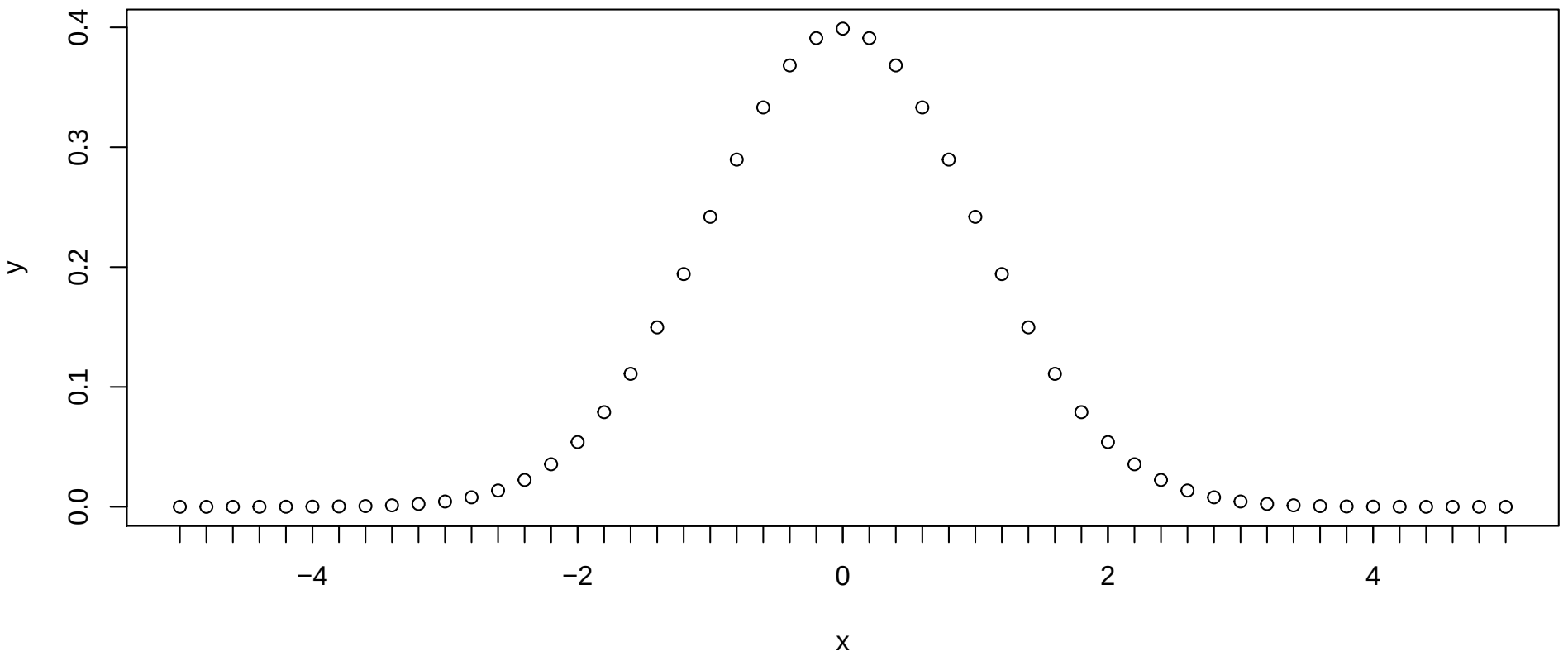
Sakysime, kad atsitiktinis dydis x turi normalųjį skirstinį, jei jo tankis

$$\varphi_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/(2\sigma^2)} \text{ for } -\infty < x < \infty; -\infty < \mu < \infty, \sigma^2 > 0$$

Sakysime, kad atsitiktinis dydis x turi standartinį normalųjį skirstinį, jeigu $\mu = 0, \sigma^2 = 1$

Normalusis skirstinys

```
x <- seq(from=-5, to=5, by=0.2)
y <- dnorm(x)
plot(x,y, type = "p")
axis(side = 1, at = x,labels = F)
```



Normalusis skirstinys

```
x <- seq(from=-5, to=5, by=0.2)
y <- pnorm(x)
plot(x,y, type = "p")
axis(side = 1, at = x,labels = F)
```