Rozkłady teoretyczne i symulacje

Statystyka I z R

Bartosz Maćkiewicz

W tej części kursu nauczymy się, w jaki sposób możemy w R korzystać z rozkładów teoretycznych. Na początek musimy przypomnieć sobie, czym właściwie są rozkłady teoretyczne.

Rozkład teoretyczny

- Opisany przez funkcję matematyczną.
- Określony przez parametry tej funkcji.

W R wbudowane są funkcje związane ze wszystkimi najważniejszymi w statystyce rozkładami teoretycznymi.

Pełną listę dostępnych rozkładów można zobaczyć, wpisując `?Distributions`.

Dla każdego rozkładu istnieją cztery funkcje:

- `d...` jak *density* np. `dnorm(x, mu, sigma)`
  - funkcja gęstości rozkładu
- `q...` jak *quantile* np. `qnorm(p, mu, sigma)`
  - funkcja kwantylowa
- `p...` jak *probability* np. `pnorm(q, mu, sigma)`
  - dystrybuanta (funkcja wyznaczająca rozkład prawdopodobieństwa)
- `r...` jak *random* np. `rnorm(n, mu, sigma)`
  - generator liczb losowych

Omówmy teraz zestaw czterech funkcji dla rozkładu normalnego. Interesują nas następujące funkcję z końcówką `norm`. Wszystkie cztery funkcje przyjmują 3 argumenty, dwa ostatnie - `mean` i `sd` określają parametry rozkładu teoretycznego, który nas interesuje.

Zobaczmy definicje z Wikipedii i spróbjmy powiedzieć sobie, co te funkcje robią.

- `dnorm`

  Funkcja gęstości prawdopodobieństwa (gęstość zmiennej losowej) – nieujemna funkcja rzeczywista, określona dla rozkładu prawdopodobieństwa, taka że całka z tej funkcji, obliczona w odpowiednich granicach, jest równa prawdopodobieństwu wystąpienia danego zdarzenia losowego. Funkcję gęstości definiuje się dla rozkładów prawdopodobieństwa jednowymiarowych i wielowymiarowych. Rozkłady mające gęstość nazywane są rozkładami ciągłymi.

Definicja ta powinna nam powiedzieć kilka rzeczy. Po pierwsze, wartość ten funkcji dla jakiejś określonej liczby nie jest prawdopodobieństwem przyjęcia przez naszą zmienną losową okreslonej wartości. W przypadku rozkładów ciągłych jest bowiem tak, że prawdopodobieństwo przyjęcia jakiejkolwiek określonej wartości wynosi 0 - możemy mówić tylko o pewnych przedziałach wartości. Funkcja `dnorm` nam będzie służyć głównie do celów pobocznych, takich jak rysowanie wykresów. Możemy bowiem uzyskać dzięki niej odpowiedź na pytanie, jaka jest gęstość prawdopodobieństwa w dowolnym punkcie i korzystając z tej informacji narysować krzywą. Dla tych z państwa mających inklinacje bardziej matematyczne - nie jest kłopotem, żeby znaleźć wzór tej funkcji (znów - na przykład na Wikipedii, ale też w podręczniku Howella).

- `pnorm`

Dystrybuanta jest definiowana jako prawdopodobieństwo tego, że zmienna X ma wartości mniejsze bądź równe x.

Jeżeli korzystali państwo z tablic statystycznych, to dystrybuanta nie jest dla Państwa nowością. R dostarcza nam wygodnej funkcji do jej obliczania. Czasami jednak interesuja nas nie wartości mniejsze lub równe x ale wartości większe od x. Oczywiście, moglibyśmy po prostu odjąć uzyskaną wartość od 1 korzystając z faktu, że prawdopodobieństwo musi się sumować do jednego, ale dla wygody i przejrzystości kodu możemy skorzystać z argumenu `lower.tail = FALSE`

- `qnorm`

  Odwrotna dystrubuanta (wygodniej myśleć - *kwantyle*).

Za pomocą dystrybuanty możemy dowiedzieć się, jak jakie jest prawdopodobieństwo przyjęcia przez naszą zmienną wartości poniżej innej określonej wartości. Za pomocą odwrotnej dystrybuanty możemy odpowiedzieć sobie na odwrotne pytanie - chcemy dowiedzieć się, poniżej jakiej wartości leżeć będzie określona cześć naszego rozkładu.

- `rnorm`

Funkcje z przedrostkiem `r...` pozwalają nam losować wartości z okreslonego parametrami rozkładu. Przy dużej liczbie losowań rozkład empiryczny tak losowanej zmiennej dążyć będzie do rozkładu teoretyznego, z którego losujemy. Tę cechę możemy wykorzystać przeprowadzając proste (i trudniejsze) eksperymenty. Możemy dowiedzieć się czegoś, używając dystrybuanty, a następnie "empirycznie" sprawdzić, czy rzeczywiście tak jest za pomocą wielokrotnego losowania. Pierwszym argumentem funkcji z przedrostkiem `r...` jest *liczba losowań* - jak duża ma być nasza próba z określonego rozkładu teoretycznego.

Wszystkie omówione tutaj zasady dotyczące funkcji R do pracy z rozkładem normalnym stosują się do pracy z innymi rozkładami. Należy jednak pamiętać o dość oczywistej sprawie - inne rozkłady teoretyczne okreslone są przez inny zestaw parametrów (niekoniecznie średnią i odchylenie standardowe). Generalnie jednak to, co powiedzieliśmy sobie do tej pory, ma zastosowanie. Ostatnią rzeczą, na którą należy zwrócić uwagę, jest fakt, że w przypdaku rozkładów dyskretnych funkcje z przedrostkiem `d...` pozwlają nam określić prawdopodobieństwo wystąpienia danego zdarzenia losowego (np. wypadnięcia n orłów przy rzucie monetą itp.)

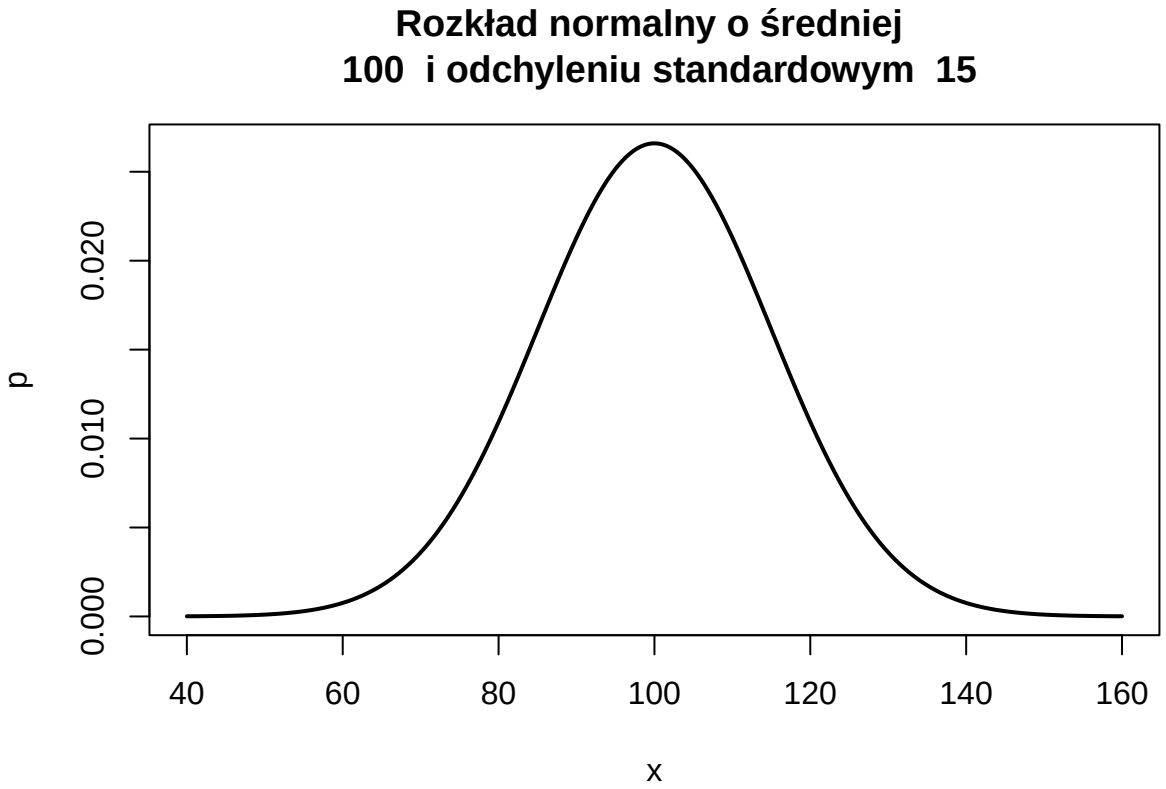
`dnorm`, `pnorm` i `qnorm` w praktyce

Rozpocznijmy od rysowania krzywej ilustrującą rozkład normalny. Żeby narysować rozkład normalny o parametrach $\mu = 100$, $\sigma = 40$, możemy użyć poniższego kodu, w którym wykorzystamy funkcję `dnorm`. Skądinąd znamy regułę trzech sigm, głoszacą, że 99,994% wartości z rozkładu normalnego znajduje się pomiedzy -4σ i 4σ (regułę czterech sigm :)). Dlatego na osi x chcemy mieć wartości od 40 do 160. Wartości na osi y obliczamy podstawiając kolejne wartości x do funkcji `dnorm`.

```
sr <- 100; odch <- 15 # parametry naszego rozkładu
x <- seq(-4, 4, length = 1000) * odch + sr # generujemy 1 000 wartości x od -4*sigma do 4*sigma

p <- dnorm(x, sr, odch) # za pomocą funkcji dnorm otrzymujemy koordynaty y naszych punktów

plot(x, p,
     type="l", # typ wykresu - liniowy
     lwd=2, # szerokość linii
     main=paste("Rozkład normalny o średniej \n",
                sr,
                " i odchyleniu standardowym ",
                odch)) # tytuł wykresu
```



Przyjrzyjmy się teraz, jak działa dystrybuanta. Chcąc dowiedzieć się, jakie jest prawdopodobieństwo, że nasza zmienna przyjmie wartość mniejszą lub równą 80 musimy posłużyć się funkcją `pnorm`.

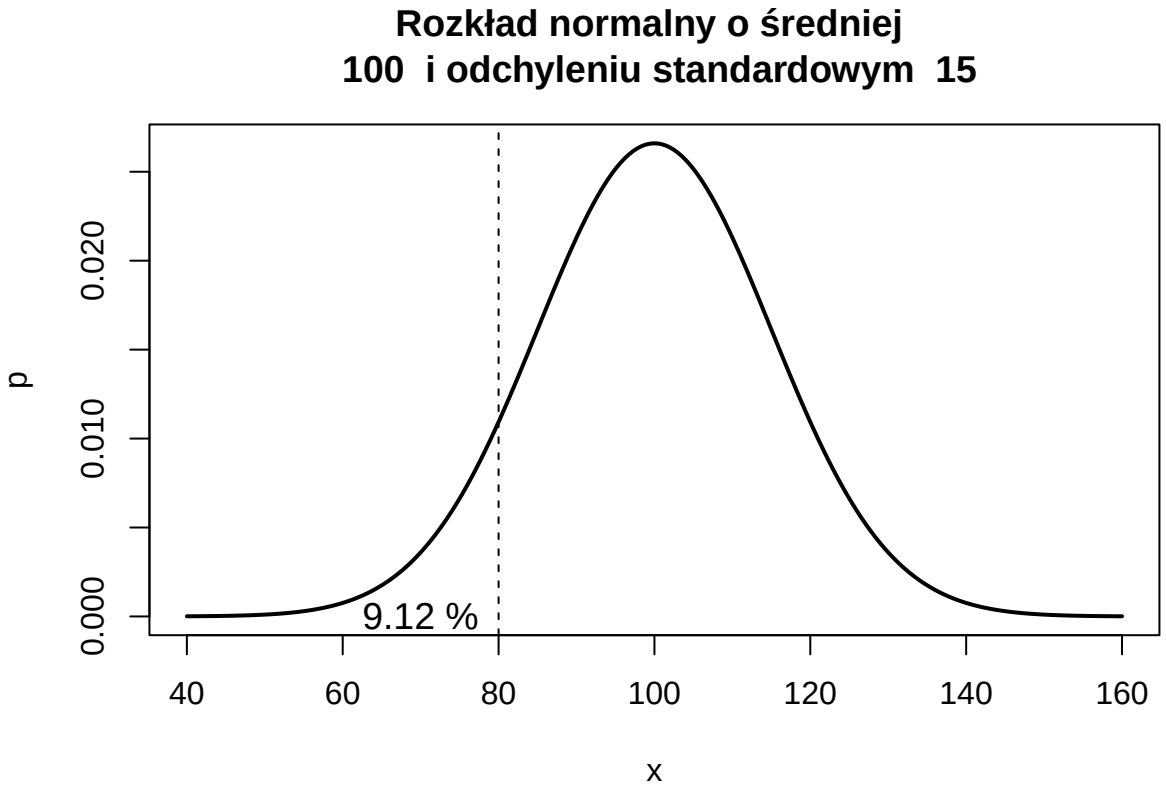
```
pnorm(80, 100, 15) # obliczamy prawdopdobieństwo X < 80
```

```
## [1] 0.09121122
```

```
plot(x, p,
     type="l", # typ liniowy wykresu
     lwd=2, # szerokość linii
     main=paste("Rozkład normalny o średniej \n", sr,
                " i odchyleniu standardowym ", odch))

abline(v=80, lty=2) # rysujemy pionową przerywaną linie przy wartości 80

# dodajemy napis ile wartości leży poniżej tej linii
text(x=70,
     y=0,
     labels=paste(round(pnorm(80, 100, 15)*100, 2),"%"),
     cex=1.2)
```



Jeżeli chcemy na przykład dowiedzieć się poniżej jakiej wartości leży 80% naszego rozkładu, musimy użyć funkcji `qnorm` (odwrotna dystrybuanta).

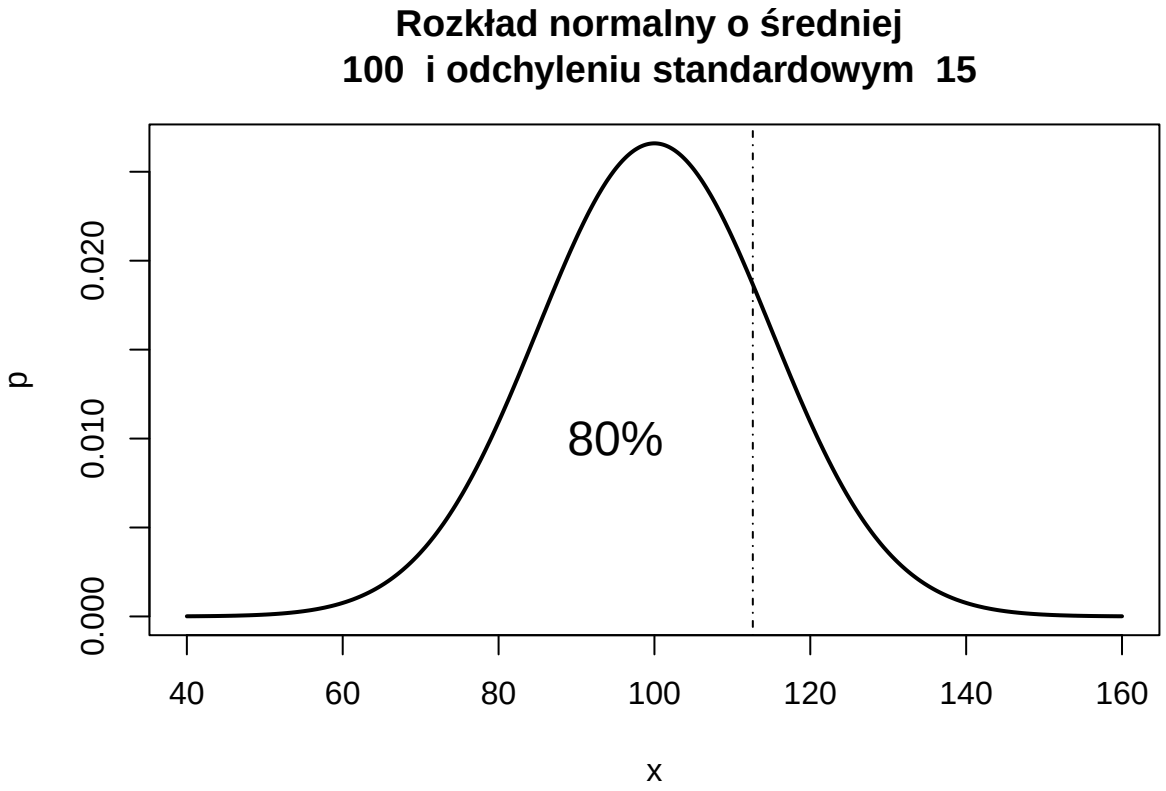
```
qnorm(.8, 100, 15) # obliczamy poniżej jakiej wartości leży 80%
```

```
## [1] 112.6243
```

```
plot(x,p,
     type="l", # typ liniowy wykresu
     lwd=2, # szerokość linii
     main=paste("Rozkład normalny o średniej \n", sr,
                " i odchyleniu standardowym ", odch)) # tytuł

# pionowa przerywana linia na wysokości 112,62 (80 percentyl)
abline(v=qnorm(.8, 100, 15), lty=4)

text(x=95, y=.01, labels="80%", cex=1.5) # podpis
```



Symulacje statystyczne (losowanie)

Po co w ogóle to robić?

Jest kilka powodów, dla których "eksperymenty" polegające na losowaniu z rozkładów toretycznych są interesujące. Pierwszy z nich jest taki, że jest to dobre narzędzie dydaktyczne - pozwala sobie uzmysłowić jak "działają" pewne zależności statystyczne. Z drugiej strony pozwalają nam dowiedzieć się o nich czegoś nowego. Bardzo dobrym przykładem jest estymowanie mocy testu statystycznego. W przypadku niektórych testów statystycznych możemy bardzo dokładnie obliczyć moc testu, ale czasami jest to trudniejsze - wówczas możemy sobie pomóc generując za pomocą R (lub innego programu) odpowiednie próby i zobaczyć, w ilu przypadkach hipoteza zerowa, która powinna być odrzucona, nie została odrzucona. Na początek jednak musimy zaznajomić się z podstawowymi narzędziami służącymi do przeprowadzania losowań, dlatego w dalszej cześci notatnika każde zadanie rozwiązywać będziemy na dwa sposoby - korzystając z rozkładów teoretycznych i estymując z losowań.

`set.seed`

1. Komputer generuje liczby pseudolosowe. Oznacza to, że tak naprawdę nie są to liczby losowe, a jedynie "udają" losowe w odpowiednio przekonujący sposób i możemy dla celów praktycznych traktować je, jako losowe liczby. Jest to spowodowane tym, że komputery ze swojej istoty losowe nie są - potrafią wykonywać algorytmy i w zasadzie nic więcej. Komputery jednak potrafią generować liczby pseudolosowe korzystając z faktu, że przy odrobinie wysiłku możemy zebrać z otoczenia pewną "dozę losowości" i nastepnie za pomocą sprytnego algorytmu w toku kolejnych jego iteracji generować liczby, które do złudzenia przypominają liczby losowe. W zależności od języka programowania i metody losowania ta doza losowości może pochodzić z różnych źródeł - z ruchów myszką użytkownika, z wahań temperatury procesora, z systemowego zegara itp. Konsekwencją tego jest, że chociaż liczby nie są naprawdę losowe (bo można je wyliczyć znając stan początkowy przy pierwszej iteracji algorytmu), to trudno to zrobić. Jest to dość skrótowe i uproszczone przedstawienie tematu, ale na potrzeby naszych zajęć powinno być

wystarczające.

2. Konsekwencją sposobu, w jaki komputery generują liczby pseudolosowe jest fakt, że możemy sami zdecydować, jaki będzie stan początkowy naszego algorytmu (ziarno, *seed*). Dzięki temu, jeżeli będzie powtarzać generowanie liczb, to za każdym razem będą takie same. Kiedy zajmujemy się kryptografią albo z jakiegoś innego powodu zależy nam na prawdziwej losowości. My, zajmując się statystyką, możemy jednak wykorzystać pseudolosowość własnych celów. Chcielibyśmy bowiem, żeby każdy, kto zna naszego *seeda* i dysponuje kodem źródłowym naszego programu mógł otrzymać dokładnie taki sam wynik. Żeby ktoś łatwo mógł zreplikować naszą symulację, używamy `set.seed` - funkcji, która ustawia ziarno na okreslona liczbę.

```
set.seed(12345) # możemy podać dowolną liczbę
```

Rzut kostką

Pierwszy przykład będzie dotyczył rzucania kostką. Na początek wykonajmy symulację 6000 rzutów kostką. Policzmy, ile było poszczególnych wyników. Aby to zrobić użyjemy funkcji `sample`:

```
sample(wektor, size, replace = T/F)
```

- `wektor` - wektor, z którego chcemy losować wartości
- `size` - liczba losowań
- `replace` - czy chcemy losować bez zwracania czy ze zwracaniem

```
kostka <- 1:6
symulowane_rzuty <- sample(kostka, # wektor, z którego losujemy wartości
                           size = 6000, # liczba losowań
                           replace = TRUE) # ze zwracaniem?
```

Policzmy, ile wyników dla każdej z sześciu możliwości otrzymaliśmy:

```
table(symulowane_rzuty)
```

```
## symulowane_rzuty
##    1    2    3    4    5    6
## 1019  981  995 1007 1002  996
```

Spodziewamy się, że otrzymamy około 1000 wystąpień każdej możliwości.

Kolejne pytanie, jakim się zajmiemy, brzmi następująco: jak często w takim eksperymencie otrzymamy więcej niż 1030 jedynek? Dość trudno obliczyć prawdopodobieństwo takiego zdarzenia, ale możemy wykorzystać symulacje, aby to prawdopodobieństwo przybliżyć.

Spróbujmy powtórzyć nasze doświadczenie 1000 razy i zobacyć, w ilu przypadkach z tego tysiąca jedynek było więcej niż 1030. Aby się tego dowiedzieć użyjmy funkcji `replicate`. Przyjmuje ona dwa argumenty - pierwszy to liczba replikacji, drugi to wyrażenie, które R ma ewaluować tyle razy, ile wynosi liczba replikacji (w przeciwieństwie do funkcji `rep`, w której R ewaluuje wyrażenie jeden raz a następnie powtarza jego wartość).

```
replicate(liczba_replikacji, {wyrażenie})
```

Jak dowiemy się, czy w poszczególnych powtórzeniach naszego eksperymentu wystąpiło więcej niż 1030 jedynek? Możemy użyć następującego wyrażenia:

```
sum(
  sample(kostka, size = 6000, replace = T) == 1 # sprawdzamy, gdzie były jedynki
  ) > 1030 # obliczamy liczbę wartości TRUE i sprawdzamy, czy jest większa niż 1030
```

```
## [1] FALSE
```

Teraz możemy to wyrażenie przekazać do funkcji `replicate`, aby powtórzyć nasze doświadczenie 1000 razy.

```
jedynki <- replicate(1000, # liczba powtórzeń naszego doświadczenia
                     {
                      sum(
                        sample(kostka, size = 6000, replace = T) == 1 # sprawdzamy, gdzie były jedynki
                        )  # obliczamy liczbę wartości TRUE i sprawdzamy, czy jest większa niż 1030
                     })

# `jedynki` sa teraz wektorem o długości 1000 z liczbą jedynek w poszczególnych replikacjach eksperymen

# ile jest takich eksperymentów, w których jedynek jest więcej niż 1030 i jaką cześć stanowią wszystkic
length(jedynki[jedynki > 1030]) / length(jedynki)
```

```
## [1] 0.145
```

```
# alternatywnie:
mean(jedynki > 1030)
```

```
## [1] 0.145
```

Rzut monetą

Zróbmy kilka ćwiczeń z prawdopodobieństwa. Rozważmy dziesięciokrotny rzut symetryczną monetą. Będziemy używać rozkładu dwumianowego. Opisują go dwa parametry - prawdopodobieństwo sukcesu oraz liczba prób:

```
...binom(..., liczba_prob, prawdopodobienstwo_sukcesu)
```

W przypadku 10 rzutów monetą będziemy więc korzystać z funkcji `dbinom`, `pbinom`, `qbinom` i `rbinom` z następującymi parametrami:

```
...binom(..., 10, 0.5)
```

Jakie jest prawdopodobieństwo wyrzucenia:

- dokładnie 4 orłów,

```
dbinom(x=4,
       size=10,
       prob=0.5)
```

```
## [1] 0.2050781
```

```
# możemy odpowiedzieć korzystając z rozkładu dwumianowego i gęstości prawdopodobieństwa

# możemy jednak zrobić to za pomocą dużej liczby losowań!

s <- rbinom(size = 10,
            n = 10000,
            prob = 0.5) # wylosujmy 10 000 wartości z tego rozkładu


length(s[s==4])/length(s) # sprawdźmy proporcję
```

```
## [1] 0.2004
```

```
# alternatywnie: mean(s == 4)
```

- co najmniej 4 orłów.

```r
# jeśli przekażemy funkcji wektor dłuższy niż jeden, to otrzymamy wektor dłuższy niż jeden
# możemy go potem zsumować
sum(
  dbinom(x=4:10,
         size=10,
         prob=0.5)
  )
```

```
## [1] 0.828125
```

```r
# możemy to samo zrobić z dystrybuanty
pbinom(3, # uwaga! w przypadku rozkładów dyskretnych musimy uważać na nierówności (czy są ostre, czy ni
       size=10,
       prob = 0.5,
       lower.tail = FALSE)
```

```
## [1] 0.828125
```

```r
# albo losując
s <- rbinom(size = 10,
            n = 10000,
            prob = 0.5)
length(s[s>=4])/length(s)
```

```
## [1] 0.8314
```

Rozkład hipergeometryczny

Zobaczmy, czy poradzimy sobie z kolejnym zdaniem.

Jako szef kampusu masz za zadanie skompletować delegację składającą się z 7 członków organizacji. Organizacja składa się z 18 kobiet oraz 15 mężczyzn. Jakie jest prawdopodobieństwo, że w losowo wybranej delegacji będzie więcej niż 4 mężczyzn?

Załózmy, że nie wiemy, z jakiego rozkładu teoretycznego chcemy skorzystać. Możemy jednak przybliżyć interesujące nas prawdopodobieństwo za pomocą symulacji. Spróbujmy najpierw "wymodelować" naszą organizację. Wiemy że znajduje się tam 15 mężczyzn i 10 kobiet, więc stwórzmy odpowiadający naszej organizacji wektor typu `character`.

```r
org <- c(rep('M', 15),
         rep('K', 18)
         ) # reprezentujemy nasza organizację jako wektor
```

Teraz mozemy przeprowadzić już symulację. Pojedyncze powtórzenie eksperymentu wyglądać będzie tak:

```r
sample(org, 7, replace = F)
```

```
## [1] "M" "K" "M" "M" "M" "K" "M"
```

Sprawdźmy, ile w naszym wylosowanym wektorze mamy mężczyzn:

```r
sum(sample(org, 7, replace = F) == "M")
```

```
## [1] 4
```

Teraz możemy już napisać kod przeprowadzający symulację. Skorzystamy ze znanej już nam funkcji 'replicate".

```r
s <- replicate(10000, # eksperyment powtarzamy 1000 razy
               {
                sum(sample(org, 7, replace = F) == "M")
               }
```

```
               )

length(s[s>4])/length(s) # obliczamy proporcję
```

```
## [1] 0.1338
```

```
# mean(s > 4)
```

Gdybyśmy akurat wiedzieli, że odpowiednim rozkładem jest tutaj rozkład hipergeometryczny, to moglibyśmy użyć odpowiednich funkcji wbudowanych w R. Zobaczmy jak blisko dokładnego prawdopodobieństwa interesującego nas zdarzenia jest nasza estymacja z symulacji.

```
# rozwiązanie używające rozkładu hipergeometrycznego
sum(dhyper(5:(15+18), 15, 18, 7)) # sumowanie gęstości
```

```
## [1] 0.1301446
```

```
phyper(4,15,18,7, lower.tail = FALSE) # dystrybuanta
```

```
## [1] 0.1301446
```

Całkiem nieźle!