LABORATORIUM 9

- Obiekty typu time series
- Wykresy serii czasowych
- Dekompozycja serii czasowych
- Funkcja autokorelacji

Obiekty typu time series

Dowolny zbiór pomiarów wykonywanych w regularnych odstępach czasu może być traktowany jako seria czasowa. W pakiecie R istnieje specjalny obiekt typu `ts` do przechowywania i operacji na seriach czasowych. Najważniejsze argumenty konstruktora `ts (data, frequency, start, ...)` to:

- `data` - wektor albo macierz (dla wielowymiarowych serii) pomiarów. Możliwe jest także użycie ramki danych, która będzie zrzutowana na macierz poprzez funkcję `data.matrix()` .
- `frequency` - liczba obserwacji w jednostce czasu. Dla danych miesięcznych `frequency = 12` , dla danych dziennych `frequency` może być równe 7 (jednostką czasu jest tydzień) lub 30 (jednostką czasu jest miesiąc) lub 365 (jednostką czasu jest rok), dla danych minutowych możemy wybierać pomiędzy 15 (kwadrans), 60 (godzina), 1440 (doba). Im dłuższy jest szereg czasowy tym większy parametr `frequency` można wybrać, ponieważ istotne jest aby szereg obejmował możliwie dużo jednostek czasu.
- `start` - czas pierwszej obserwacji. Albo pojedyncza liczba albo wektor dwóch liczb, gdzie druga musi być całkowita, np. `c(rok, miesiąc)` , `c(rok, kwartal)` , `c(miesiac, dzien)` .

```
# Liczba urodzeń w Nowym Jorku od stycznia 1946 do grudnia 1959 roku
nybirths <- read.table ("nybirths.dat")
nybirths.ts <- ts (nybirths, frequency = 12, start = c(1946,1))
nybirths.ts
```

```
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 1946 26.663 23.598 26.931 24.740 25.806 24.364 24.477 23.901 23.175 23.227
## 1947 21.439 21.089 23.709 21.669 21.752 20.761 23.479 23.824 23.105 23.110
## 1948 21.937 20.035 23.590 21.672 22.222 22.123 23.950 23.504 22.238 23.142
## 1949 21.548 20.000 22.424 20.615 21.761 22.874 24.104 23.748 23.262 22.907
## 1950 22.604 20.894 24.677 23.673 25.320 23.583 24.671 24.454 24.122 24.252
## 1951 23.287 23.049 25.076 24.037 24.430 24.667 26.451 25.618 25.014 25.110
## 1952 23.798 22.270 24.775 22.646 23.988 24.737 26.276 25.816 25.210 25.199
## 1953 24.364 22.644 25.565 24.062 25.431 24.635 27.009 26.606 26.268 26.462
## 1954 24.657 23.304 26.982 26.199 27.210 26.122 26.706 26.878 26.152 26.379
## 1955 24.990 24.239 26.721 23.475 24.767 26.219 28.361 28.599 27.914 27.784
## 1956 26.217 24.218 27.914 26.975 28.527 27.139 28.982 28.169 28.056 29.136
## 1957 26.589 24.848 27.543 26.896 28.878 27.390 28.065 28.141 29.048 28.484
## 1958 27.132 24.924 28.963 26.589 27.931 28.009 29.229 28.759 28.405 27.945
## 1959 26.076 25.286 27.660 25.951 26.398 25.565 28.865 30.000 29.261 29.012
##         Nov    Dec
## 1946 21.672 21.870
## 1947 21.759 22.073
## 1948 21.059 21.573
## 1949 21.519 22.025
## 1950 22.084 22.991
## 1951 22.964 23.981
## 1952 23.162 24.707
## 1953 25.246 25.180
## 1954 24.712 25.688
## 1955 25.693 26.881
## 1956 26.291 26.987
## 1957 26.634 27.735
## 1958 25.912 26.619
## 1959 26.992 27.897
```

```
# Dane pobrane w 15-minutowych oknach z serwisu Twitter podczas Olimpiady w Londynie w 201
    2 roku.
twitter <- read.table ("cyber.dat", header = TRUE)
head (twitter)
```

```
##   t       date     time comments     neg    pos    emo
## 1 1 2012-07-11 16:44:00      330 -1.7303 1.9970 0.2667
## 2 2 2012-07-11 16:59:00      292 -1.7808 2.0479 0.2671
## 3 3 2012-07-11 17:14:00      373 -1.7641 2.0080 0.2440
## 4 4 2012-07-11 17:29:00      372 -1.9167 1.9704 0.0538
## 5 5 2012-07-11 17:44:00      359 -1.7855 2.0084 0.2228
## 6 6 2012-07-11 17:59:00      351 -1.7892 1.9772 0.1880
```

```
twitter.comments.ts <- ts (twitter$comments, frequency = 96)
head (twitter.comments.ts)
```

```
## [1] 330 292 373 372 359 351
```

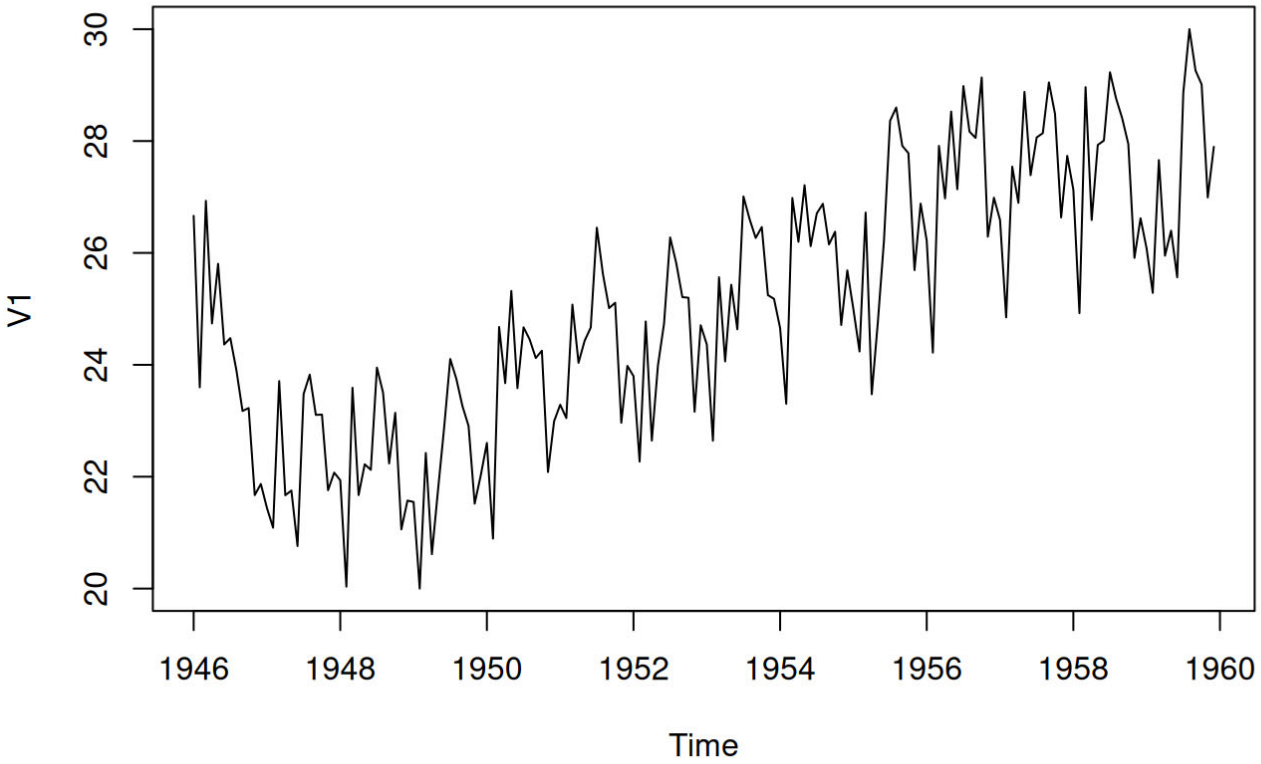
```
# Wielowymiarowa seria czasowa
twitter.mts <- ts (twitter[,4:7], frequency = 96)
head (twitter.mts)
```

```
##      comments     neg    pos    emo
## [1,]      330 -1.7303 1.9970 0.2667
## [2,]      292 -1.7808 2.0479 0.2671
## [3,]      373 -1.7641 2.0080 0.2440
## [4,]      372 -1.9167 1.9704 0.0538
## [5,]      359 -1.7855 2.0084 0.2228
## [6,]      351 -1.7892 1.9772 0.1880
```

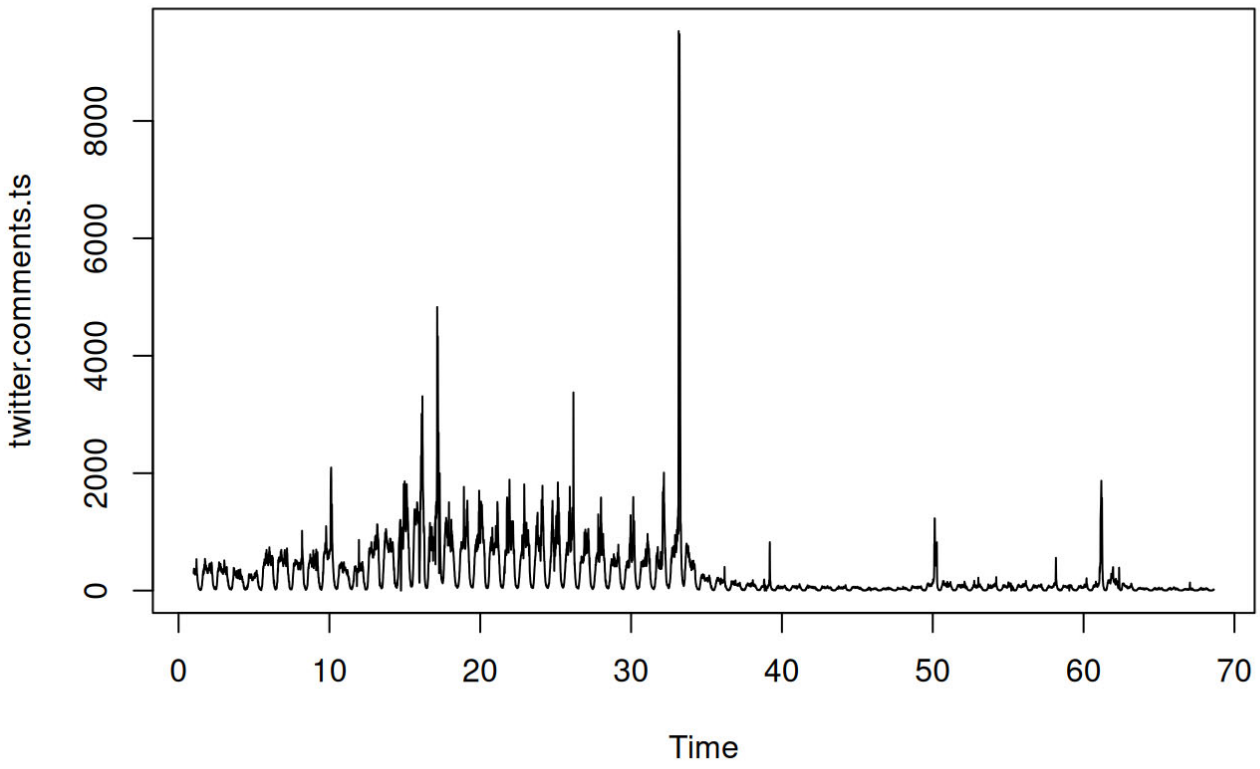
Wykresy serii czasowych

Przy pomocy funkcji `plot.ts()` można łatwo wykonać wykresy serii czasowych z odpowiednimi oznaczeniami na osi x.

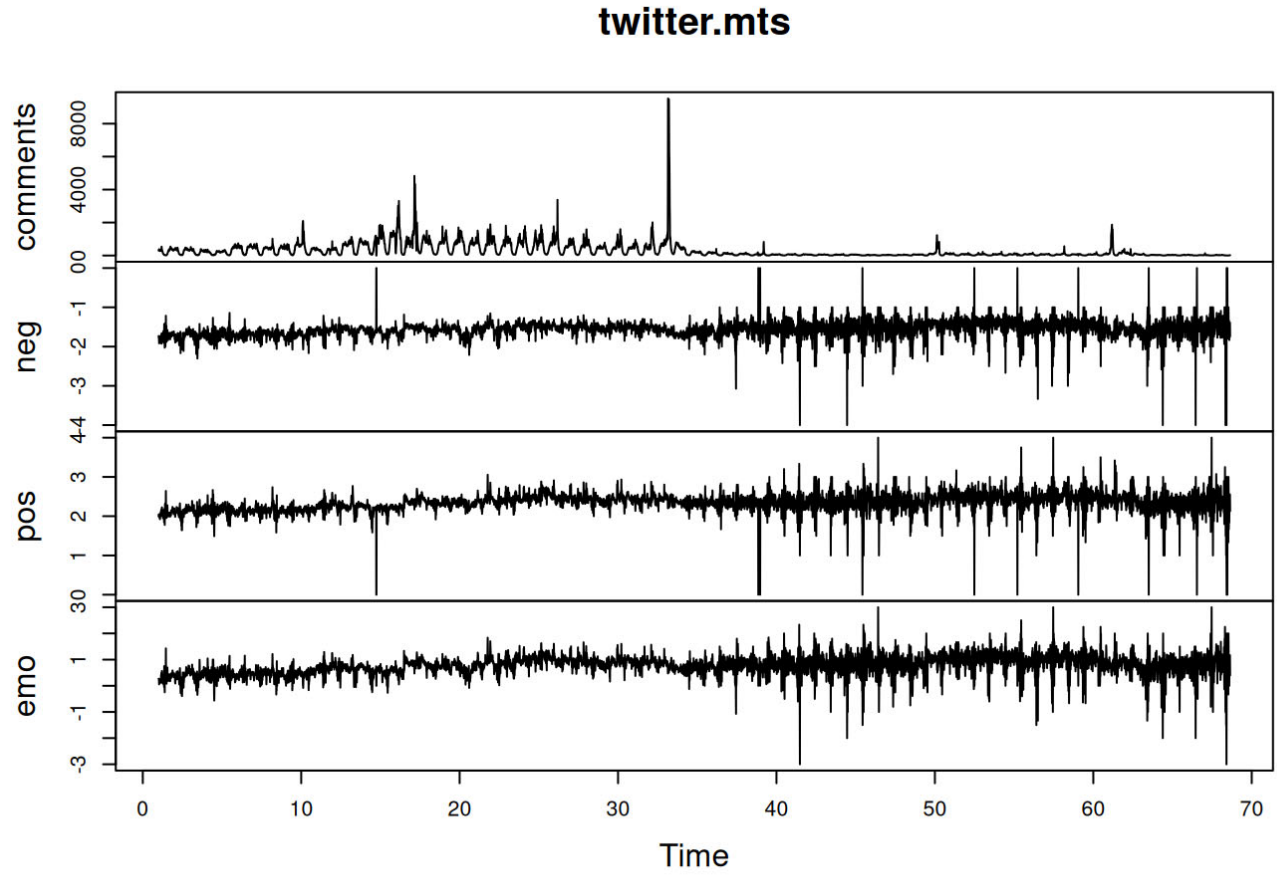
```
plot.ts (nybirths.ts)
```




```
plot.ts (twitter.comments.ts)
```

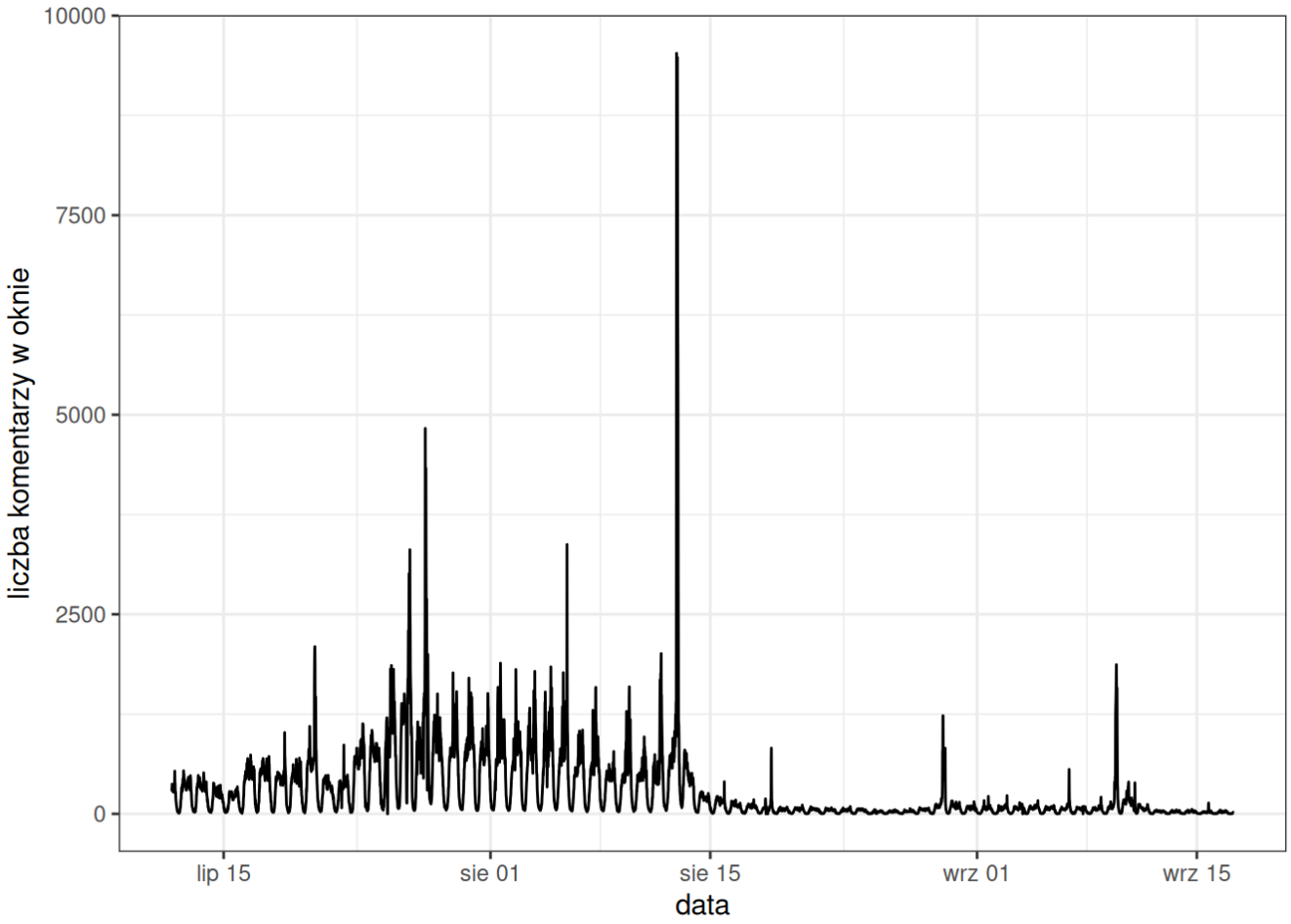



```
plot.ts (twitter.mts)
```

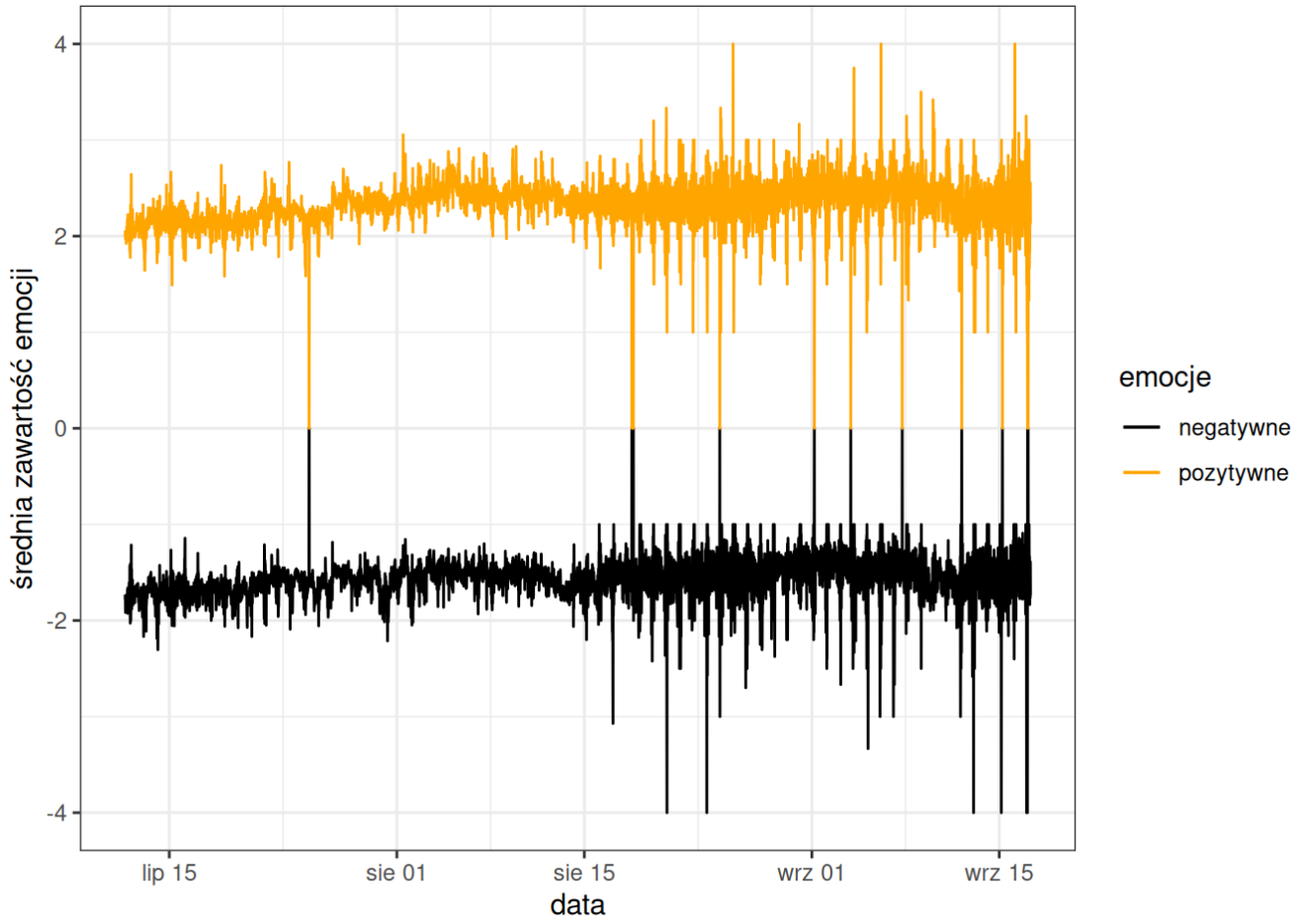


Funkcja `plot.ts()` charakteryzuje się łatwością użycia i dlatego najlepiej nadaje się do szybkich podglądów serii czasowych. Do wykonania bardziej eleganckich wykresów zaleca się użycia pakietu `ggplot2`. Odpowiedni wygląd osi x można uzyskać poprzez konwersję zmiennej typu `character` na obiekt typu `Date`. Wykorzystamy do tego funkcję `as_datetime()` z biblioteki `lubridate`.

```
library(ggplot2)
library(dplyr)
library(magrittr)
library(lubridate)

twitter %<>%
  mutate (date.time = as_datetime (paste (date, time),
                                   format = "%Y-%m-%d %H:%M:%S"))

g <- ggplot(twitter, aes (x = date.time))

g + theme_bw() +
  geom_line (aes(y = comments)) +
  labs (x = "data", y = "liczba komentarzy w oknie")
```




```
g + theme_bw() +
  geom_line (aes(y = neg, color = "negatywne")) +
  geom_line (aes(y = pos, color = "pozytywne")) +
  scale_color_manual (values = c("black","orange")) +
  labs (x = "data", y = "średnia zawartość emocji", color = "emocje")
```



Dekompozycja serii czasowych

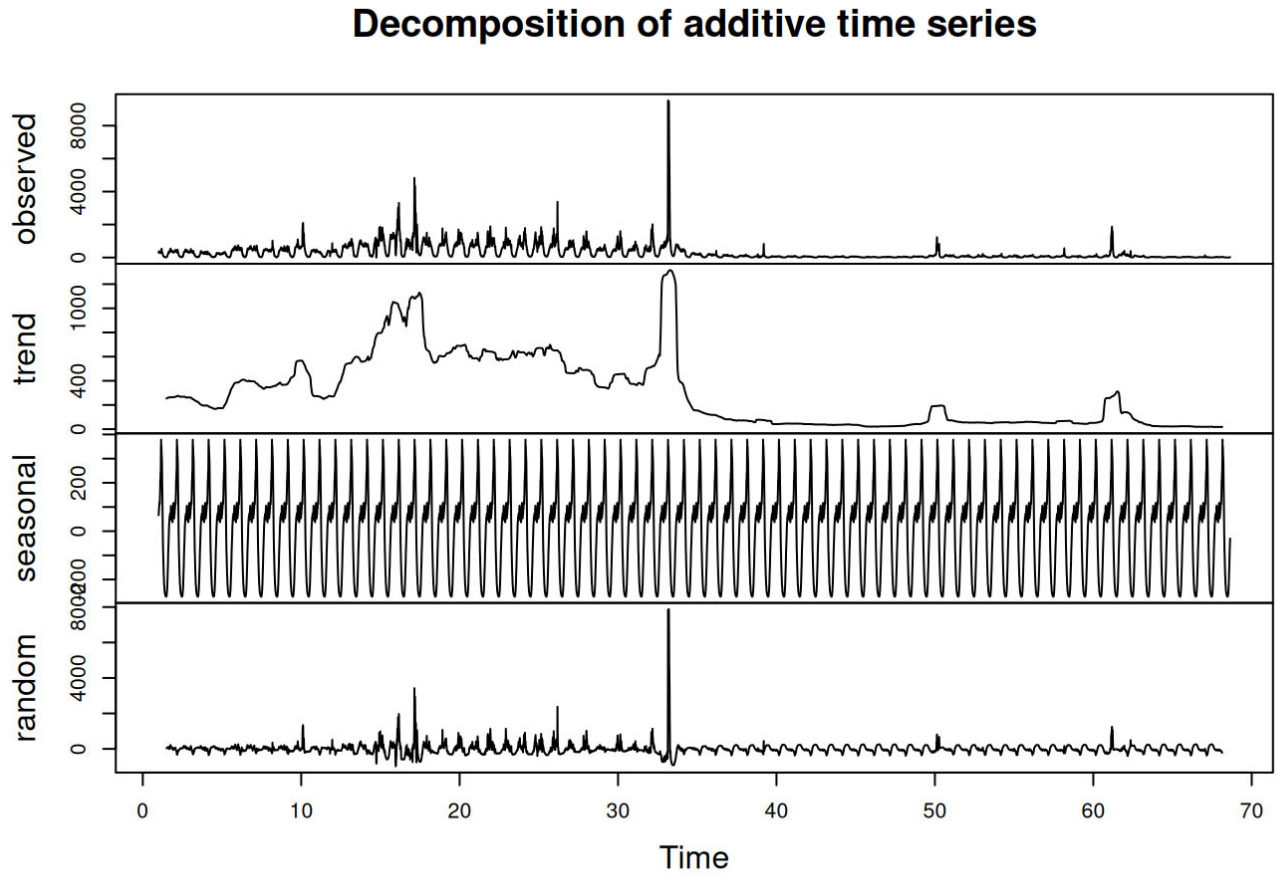
Każda seria czasowa $Y(t)$ może być wyrażona jako suma lub iloczyn trzech składowych:

- trendu $T(t)$,
- periodyczną (sezonową) $S(t)$,
- szumu lub błędu, czyli losową $\epsilon(t)$.

Dla *addytywnych* szeregów czasowych zachodzi $Y(t) = T(t) + S(t) + \epsilon(t)$, podczas gdy dla *multiplikatywnych* $Y(t) = T(t) \cdot S(t) \cdot \epsilon(t)$. Mulitplikatywny szereg czasowy może zostać zamieniony na addytywny przez zlogarytmowanie go, tzn. $\log Y(t) = \log T(t) + \log S(t) + \log \epsilon(t)$.

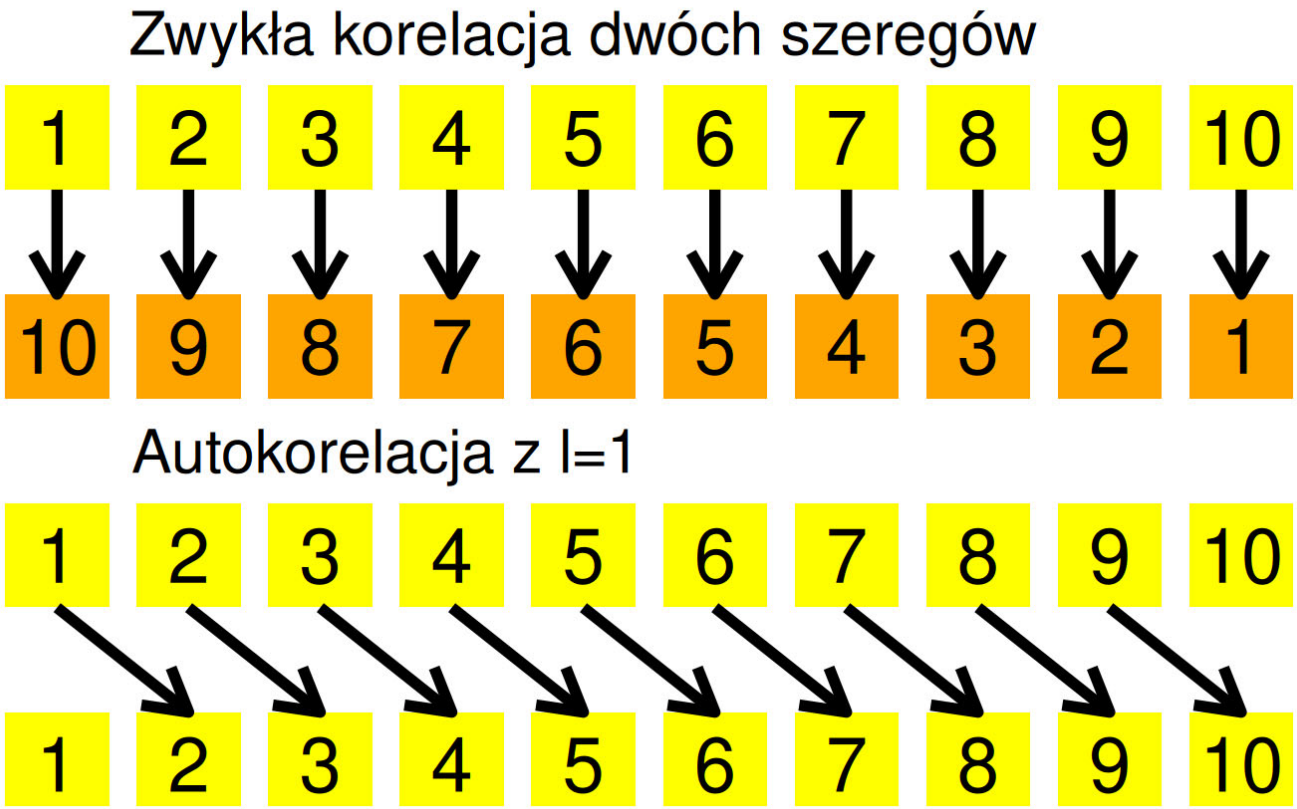
Funkcja `decompose()` umożliwia dekompozycje serii czasowej na części związane z trendem, periodycznością i szumem. Można ją stosować zarówno dla addytywnych jak i multiplikatywnych szeregów czasowych. Najważniejzym argumentem funkcji jest obiekt klasy `ts` z poprawnie określoną częstotliwością. Wynikiem działania funkcji `decompose` jest lista, której elementami są: `$x` - oryginalny sygnał, `$seasonal` - składowa periodyczna, `$trend` - trend, `$random` - część losowa, oraz `$figure` - trójpanelowy wykres, który można podejrzeć przy pomocy funkcji `plot()` .

```
comments.comp <- decompose (twitter.comments.ts)
plot(comments.comp)
```



Funkcja autokorelacji

Za pomocą funkcji `acf()` otrzymamy wartości (i domyślny wykres) funkcji autokorelacji szeregu czasowego. Autokorelacja jest po prostu współczynnikiem korelacji wyznaczonym dla tego samego szeregu, tyle że przesuniętego o pewną wartość:

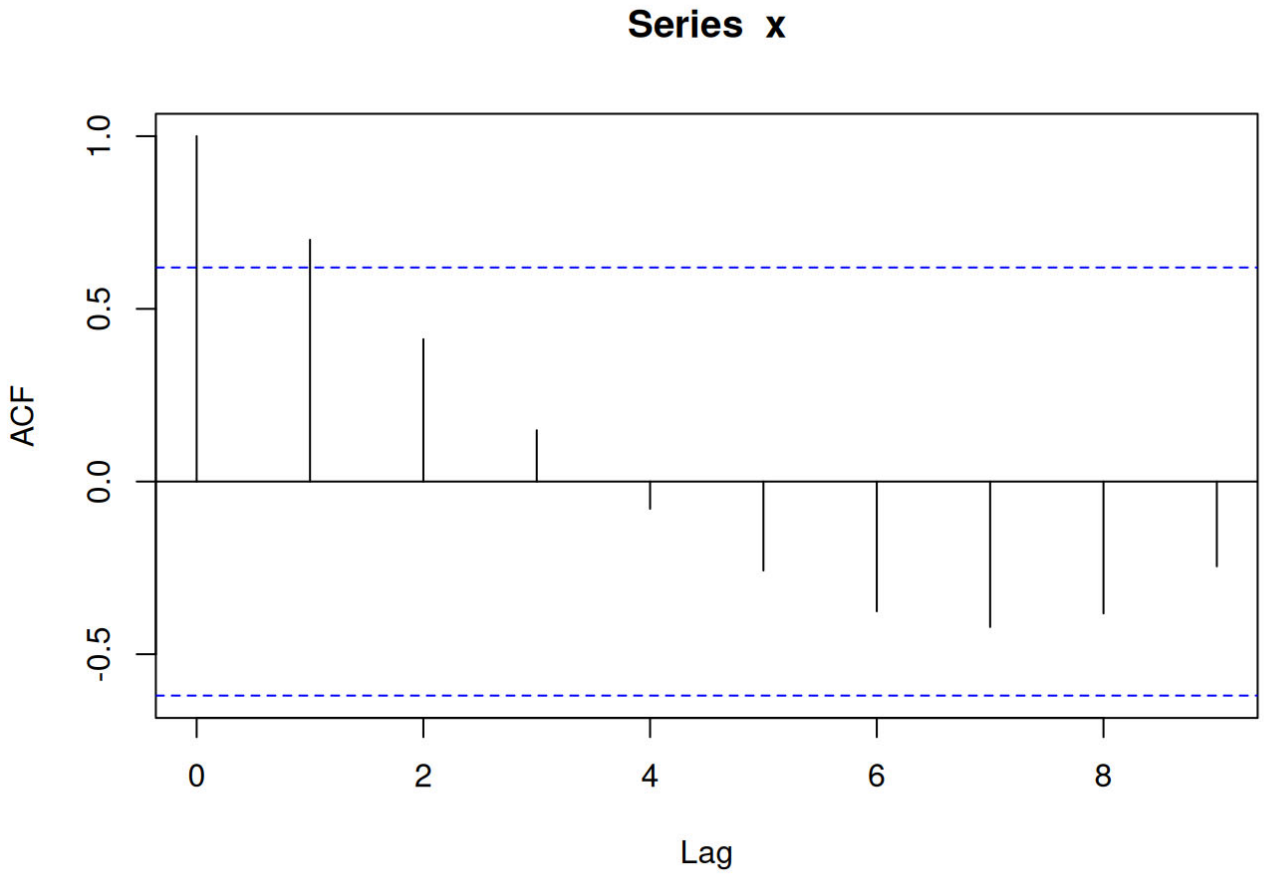


Wywołanie funkcji `acf()` automatycznie tworzy wykresy funkcji autokorelacji. Warto przy tym zwrócic uwagę, że funkcja wyznacza autokrelację korzystając z następującego wzoru (przy założeniu, że nasz szereg X ma rozmiar N a l to przesunięcie):

$$r(l) = \frac{\sum_{t=1}^{t=N-l} (X_t - \langle X \rangle)(X_{t+l} - \langle X \rangle)}{\sum_{t=1}^{t=N} (X_t - \langle X \rangle)^2}$$

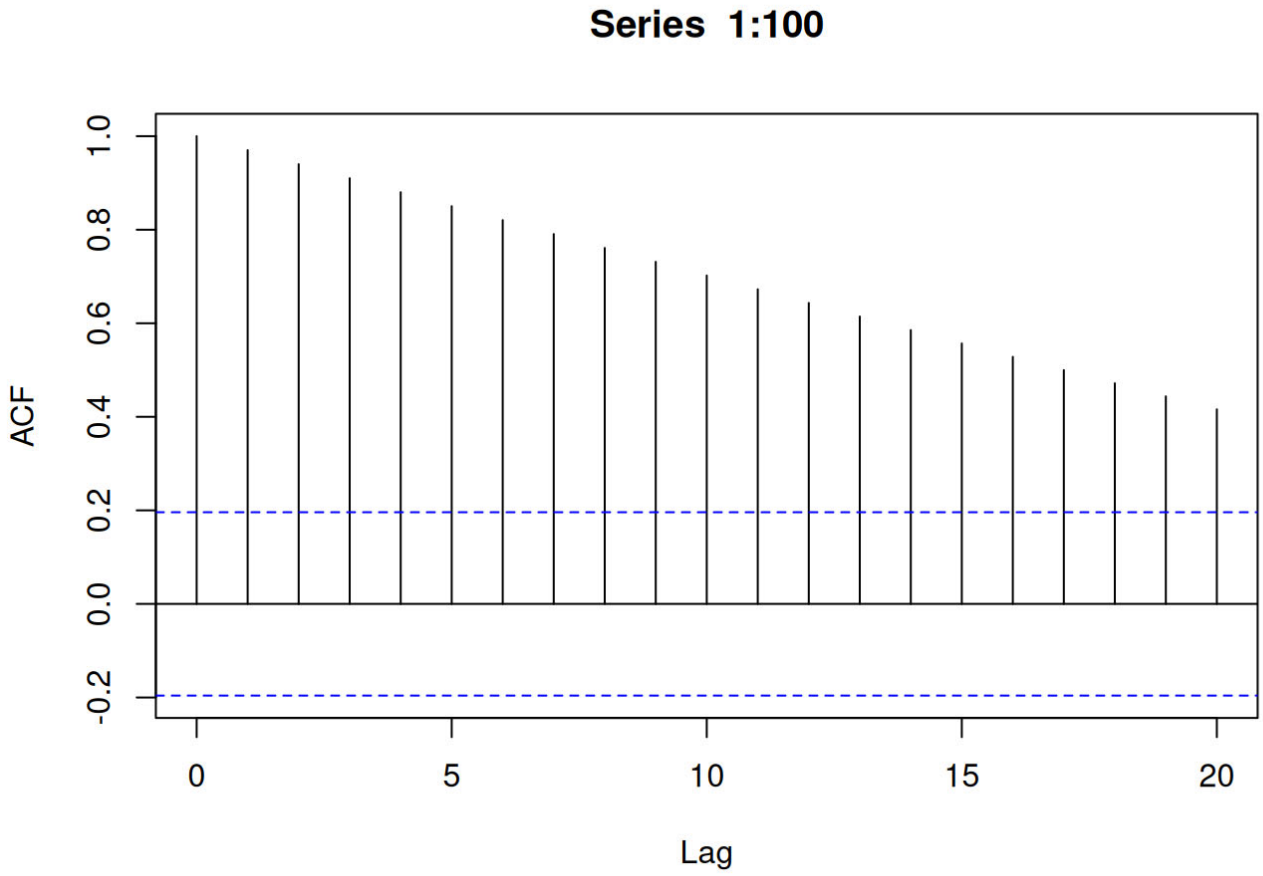
W efekcie np. dla szeregu $x = [1, 2, \ldots, 10]$ dostajemy ten pozornie dziwny wykres:

```
x <- 1:10
acf(x)
```

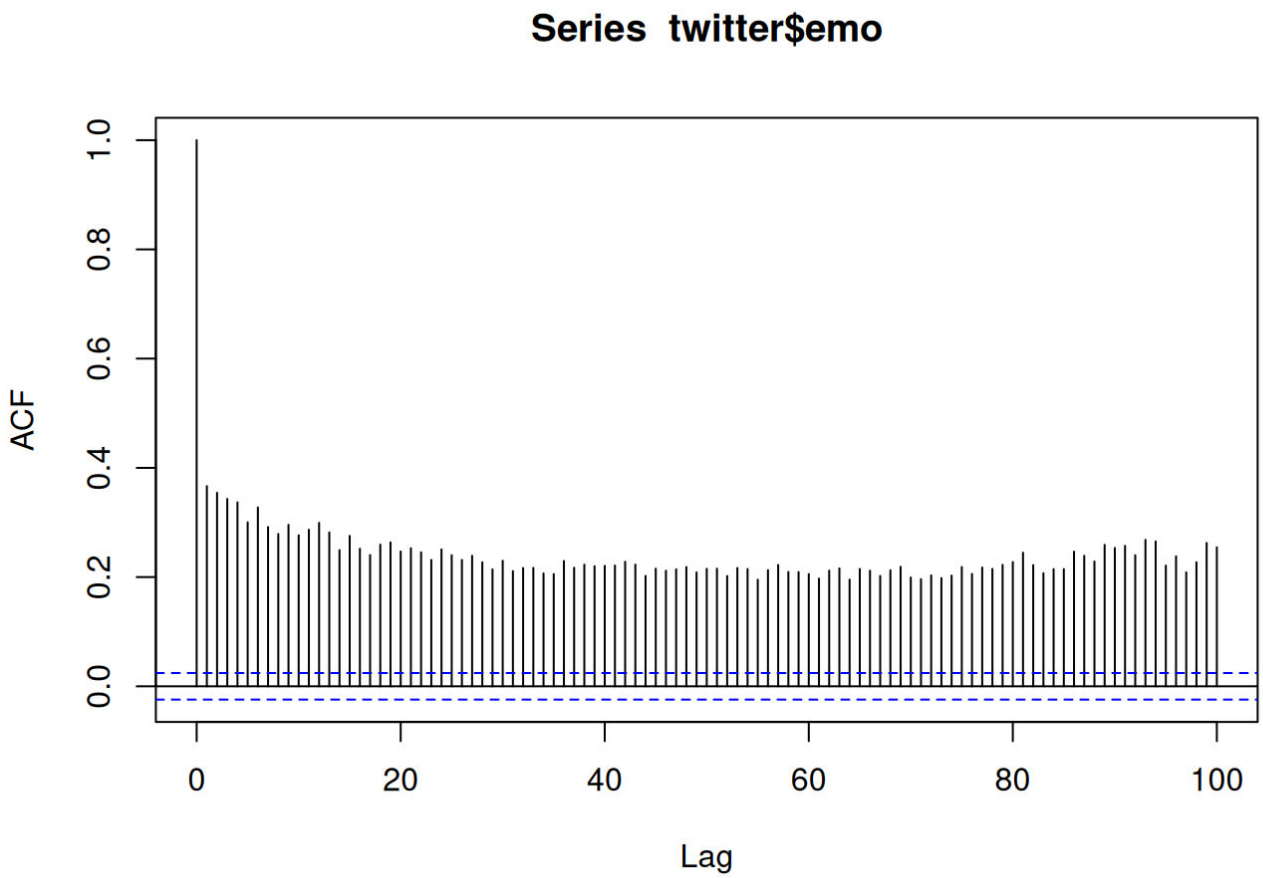


Oczywiście, w przypadku, gdy szereg jest dość długi takich problemów już nie ma.

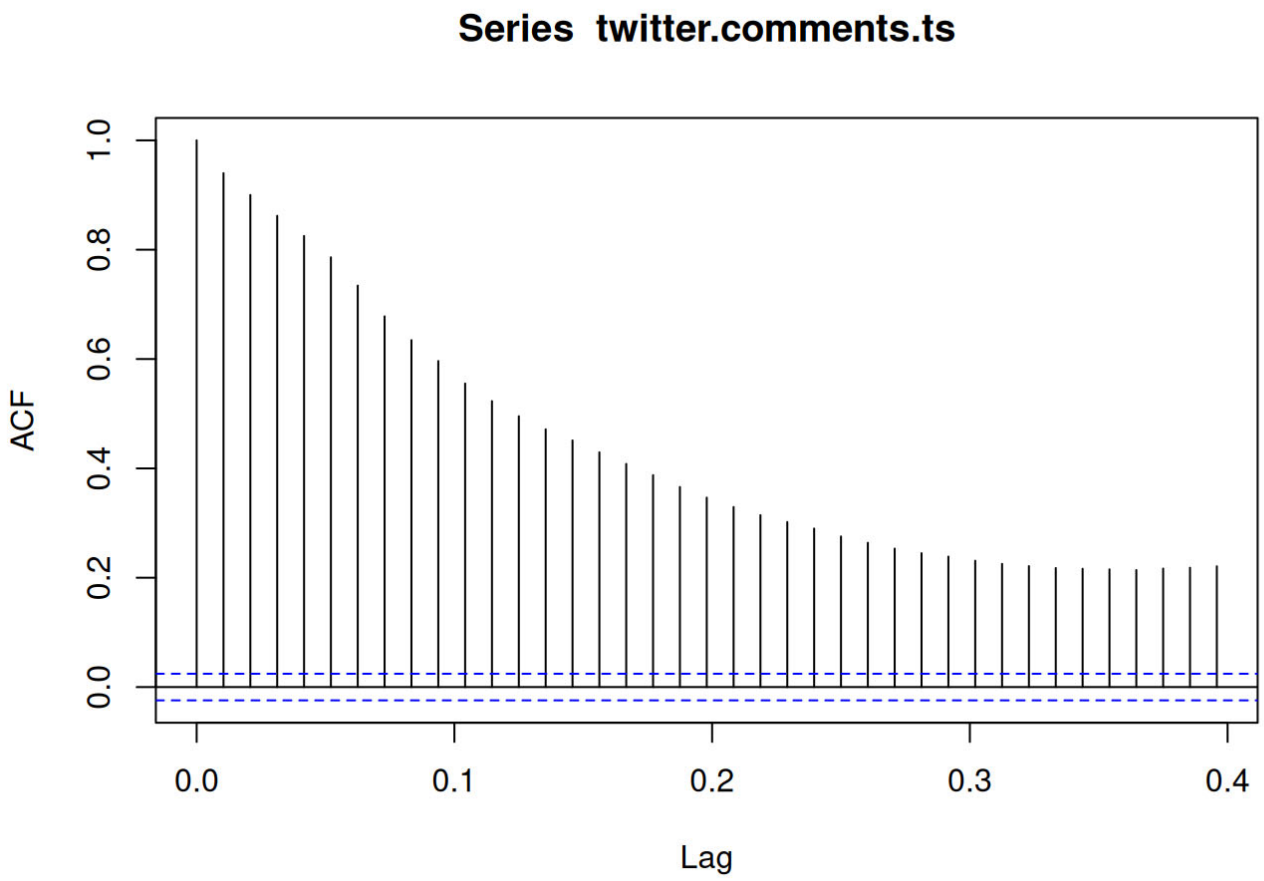
```
acf (1:100)
```




```
acf (twitter$emo, lag.max = 100)
```

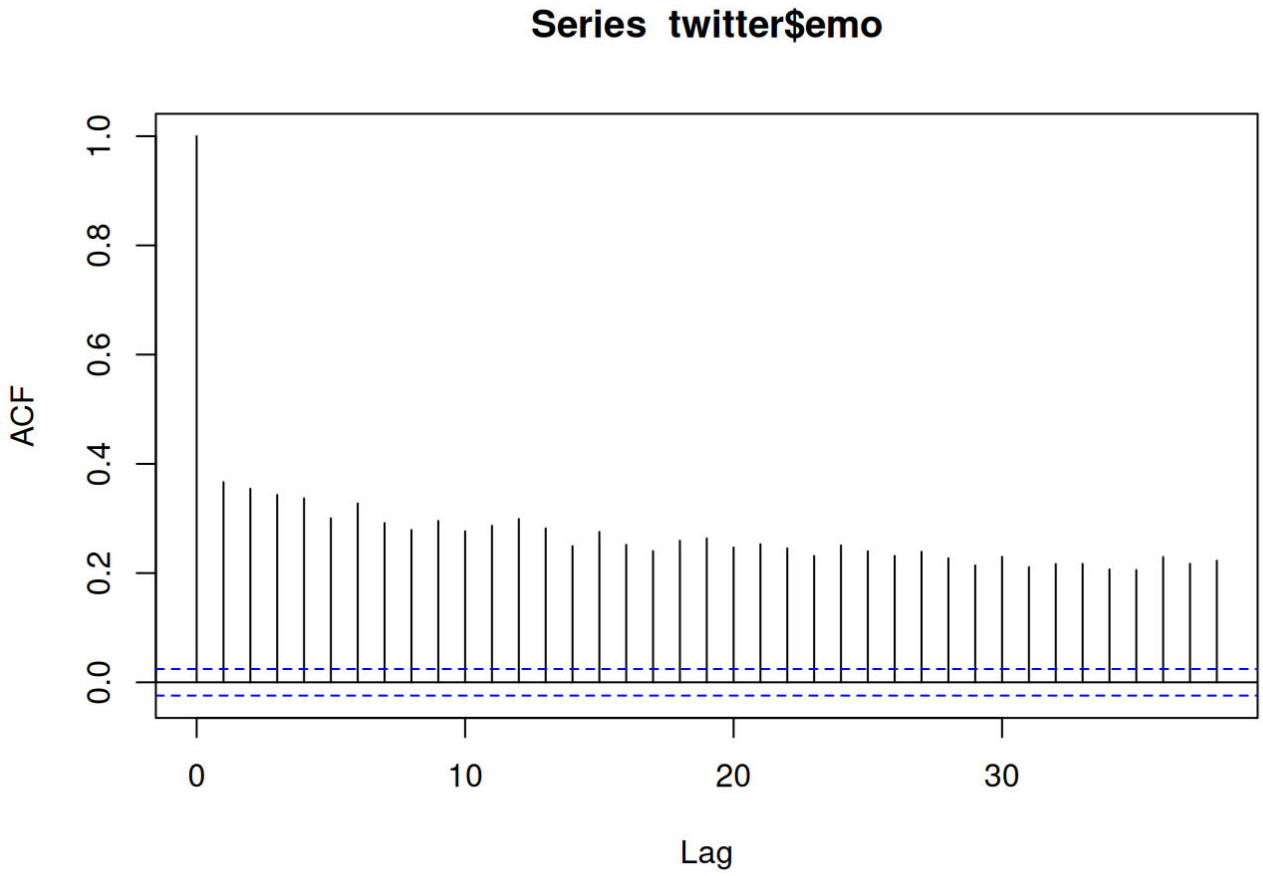


```
acf (twitter.comments.ts)
```

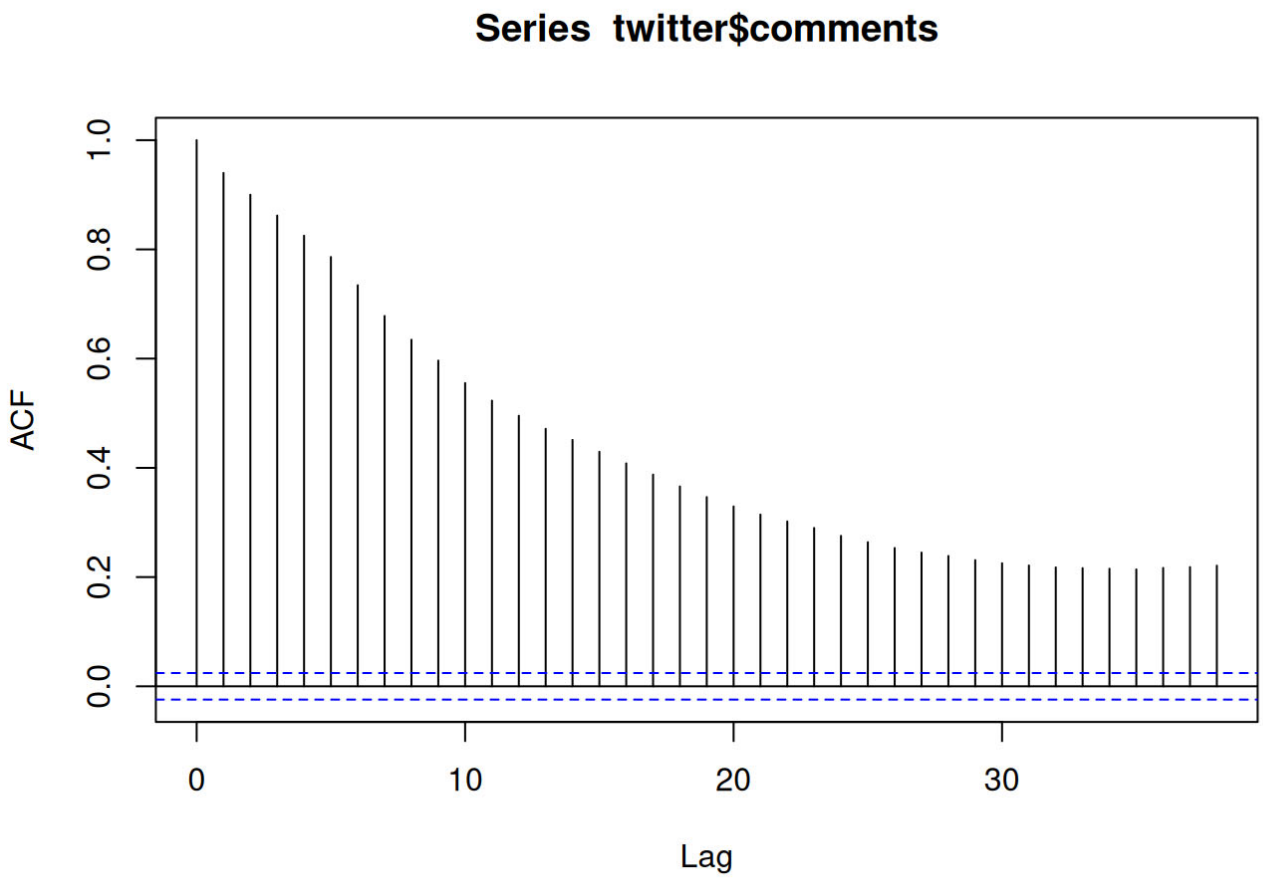


W celu porównania wykresów funkcji autokorelacji odwołujemy się do poszczególnych składowych otrzymanych zmiennych (`acf` , `lag`).

```
emo.acf <- acf (twitter$emo)
```



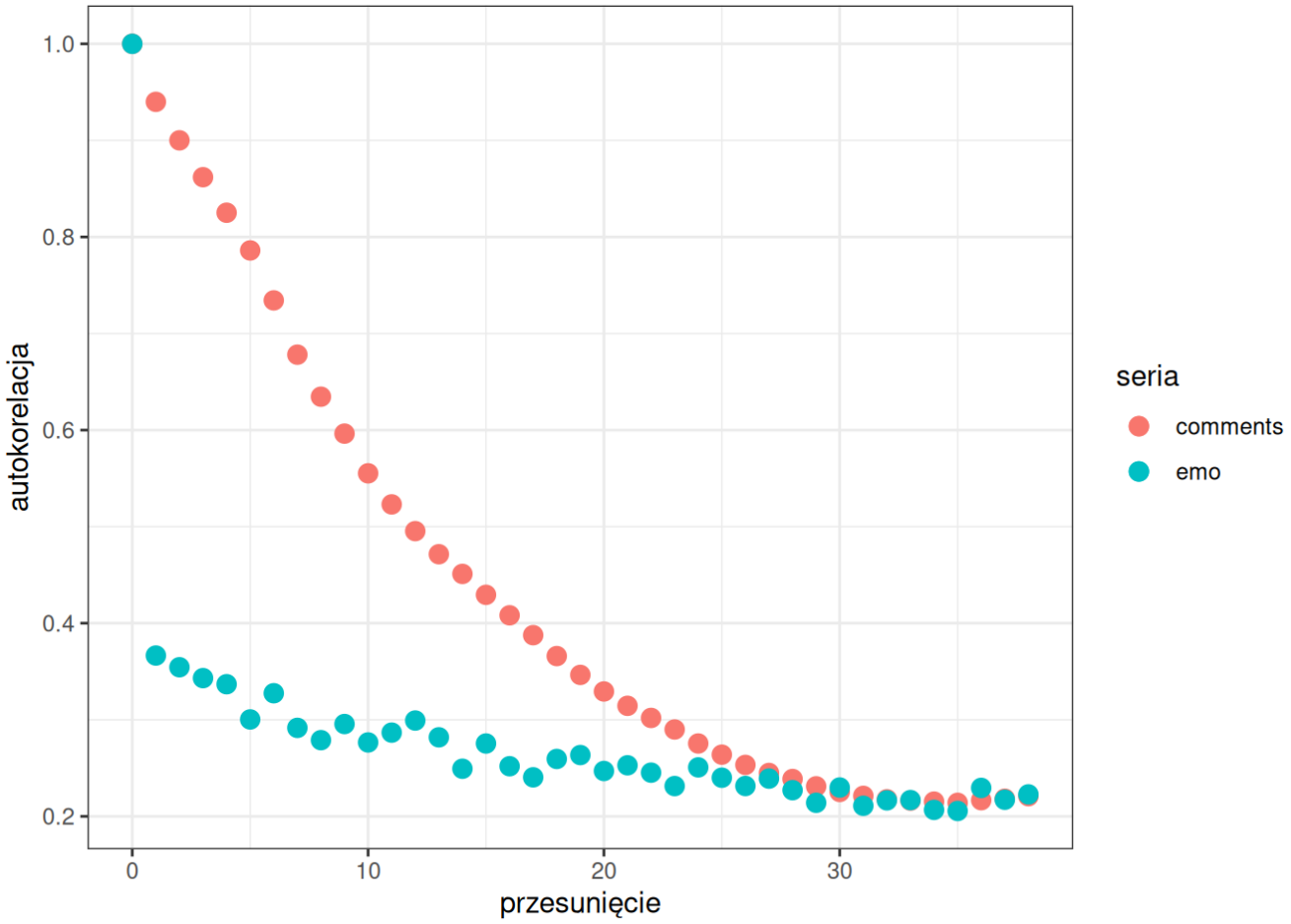

```
comments.acf <- acf (twitter$comments)
```



```
acf.df <- data.frame (lag = emo.acf$lag, comments = comments.acf$acf, emo = emo.acf$acf)
acf.df
```

```
##    lag  comments       emo
## 1    0 1.0000000 1.0000000
## 2    1 0.9399225 0.3665214
## 3    2 0.9000480 0.3543908
## 4    3 0.8618741 0.3431365
## 5    4 0.8250931 0.3368175
## 6    5 0.7860484 0.3003519
## 7    6 0.7343220 0.3274854
## 8    7 0.6781029 0.2916379
## 9    8 0.6345612 0.2789155
## 10   9 0.5963326 0.2956152
## 11  10 0.5552265 0.2765411
## 12  11 0.5230479 0.2867227
## 13  12 0.4953809 0.2992242
## 14  13 0.4714417 0.2818484
## 15  14 0.4510583 0.2493000
## 16  15 0.4293892 0.2754497
## 17  16 0.4081743 0.2518896
## 18  17 0.3876031 0.2403999
## 19  18 0.3659647 0.2594402
## 20  19 0.3464604 0.2634851
## 21  20 0.3293129 0.2469580
## 22  21 0.3144594 0.2528576
## 23  22 0.3019226 0.2452463
## 24  23 0.2899641 0.2313643
## 25  24 0.2754574 0.2506975
## 26  25 0.2638865 0.2399968
## 27  26 0.2532810 0.2314394
## 28  27 0.2449191 0.2390851
## 29  28 0.2386572 0.2270953
## 30  29 0.2310675 0.2140408
## 31  30 0.2252918 0.2298475
## 32  31 0.2212231 0.2109398
## 33  32 0.2176836 0.2165425
## 34  33 0.2163366 0.2168547
## 35  34 0.2152734 0.2066870
## 36  35 0.2140204 0.2055577
## 37  36 0.2167229 0.2294540
## 38  37 0.2182613 0.2171979
## 39  38 0.2208866 0.2227386
```

```
ggplot (acf.df, aes (x = lag)) +
  geom_point (aes (y = comments, color="comments"), size=3) +
  geom_point (aes (y = emo, color="emo"), size=3) +
  labs (x = "przesunięcie", y = "autokorelacja", color = "seria") +
  theme_bw()
```



W przypadku, gdy chcemy obliczyć korelację wzajemną dwóch szeregów z przesuniętymi wartościami, używamy funkcji `ccf()`.


```
ccf (twitter$comments, twitter$emo)
```