FORECASTING AND PREDICTIVE MODELING

LECTURE 6

Lect. PhD. Oneț-Marian Zsuzsanna

Babeș - Bolyai University
Computer Science and Mathematics Faculty

2024 - 2025

In Lecture 5

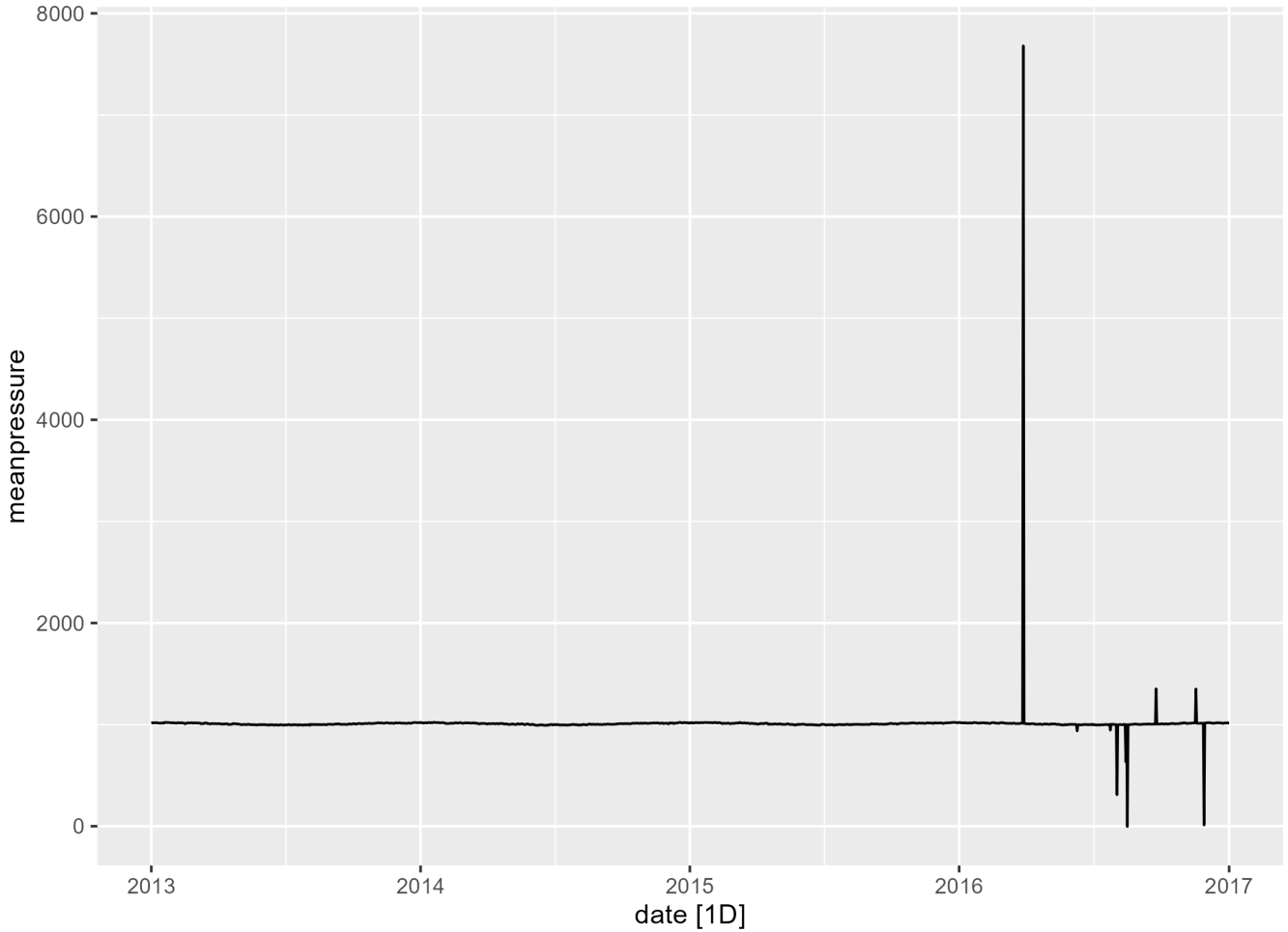
- Time series transformations
- Time series decomposition
  - Examples
  - Moving average

Today

- Time series decomposition
- Time series features

- Let us look at another example of computing the moving average.
- We will consider (part of) the Delhi Climate time series, which was used in Seminar 2.
- It has 1462 daily observations about meantemperature, humidity, wind_speed and meanpressure for Delhi.
- We will use the *meanpressure* time series. Let us visualize it:

```
climate <- read_csv("DailyDelhiClimate.csv")
climate
climate_ts <- climate |> as_tsibble(index = date)
climate_ts |> autoplot(meanpressure)
```



- What do you observe in the plot?

- In the year 2016, the *meanpressure* time series contains several outlier values, the largest one (and chronologically the first one) having a value of over 7000, while most values are around 1000.

- To see that part better, we will do two things:

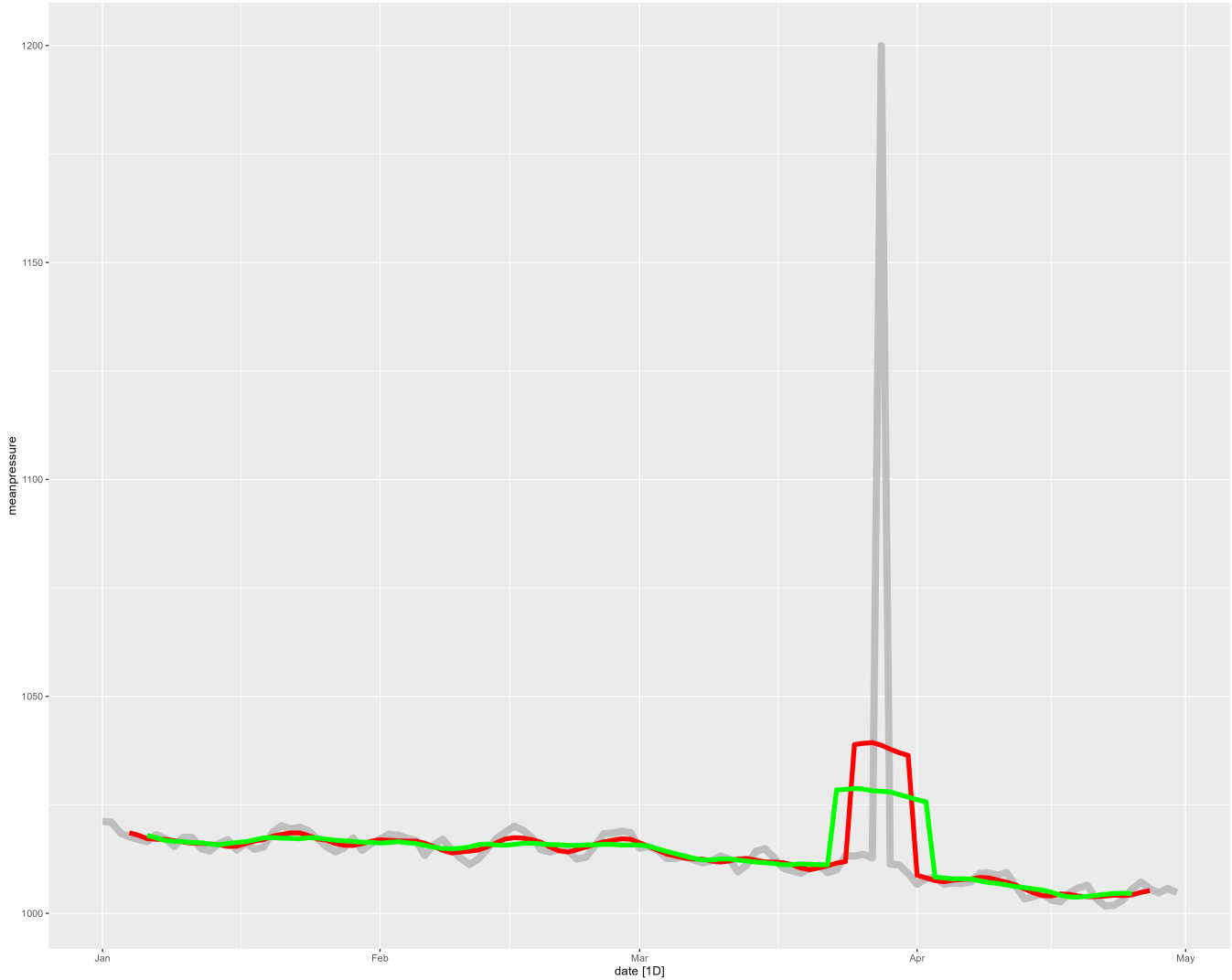
  - Change the maximum outlier to 1200, to be able to see things better on the plot

  - Consider only the first 4 months of 2016 (this eliminates the other outliers as well)

```
position <- which(climate_ts$meanpressure == max(climate_ts$meanpressure))
position

climate_ts$meanpressure[1183] <- 1200

climate_ts_short <- climate_ts |> filter(year(date) == 2016, month(date) < 5)
climate_ts_short |> autoplot(meanpressure)
```

- The data does not have seasonalities (can be checked with seasonal subseries plot) so we will compute a 7-MA and an 11-MA on it and plot them together with the original data.

```
climate_ts_short <- climate_ts_short |>
  mutate(MA7 = slider::slide_dbl(meanpressure, mean, .before = 3, .after = 3, .
      complete = TRUE)) |>
  mutate(MA11 = slider::slide_dbl(meanpressure, mean, .before = 5, .after = 5, .
      complete = TRUE))

climate_ts_short |> autoplot(meanpressure, color = "grey", size = 3) +
  autolayer(climate_ts_short, MA7, color = "red", size = 2) +
  autolayer(climate_ts_short, MA11, color = "green", size = 2)
```



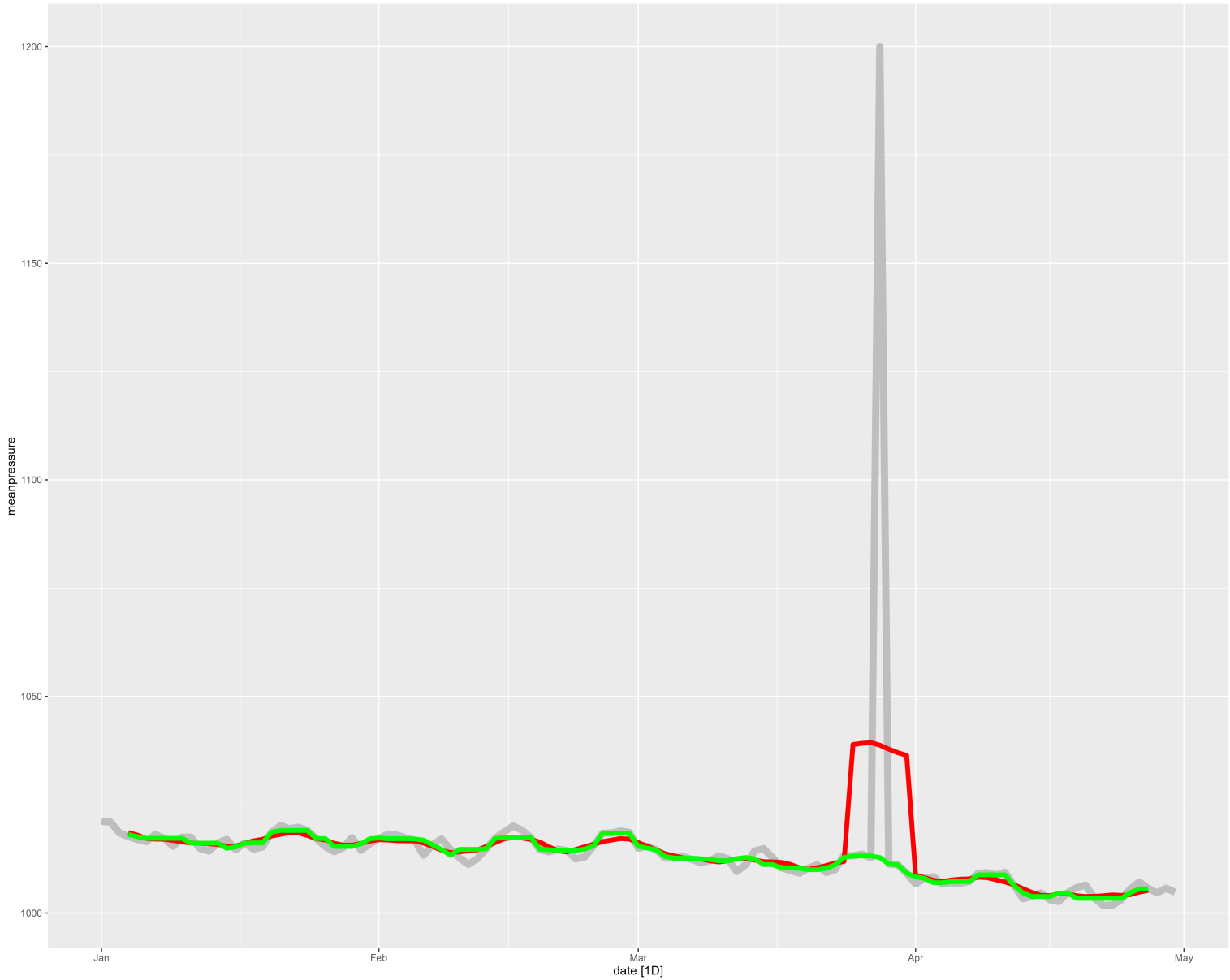
- What do you notice on the plots?

Moving median

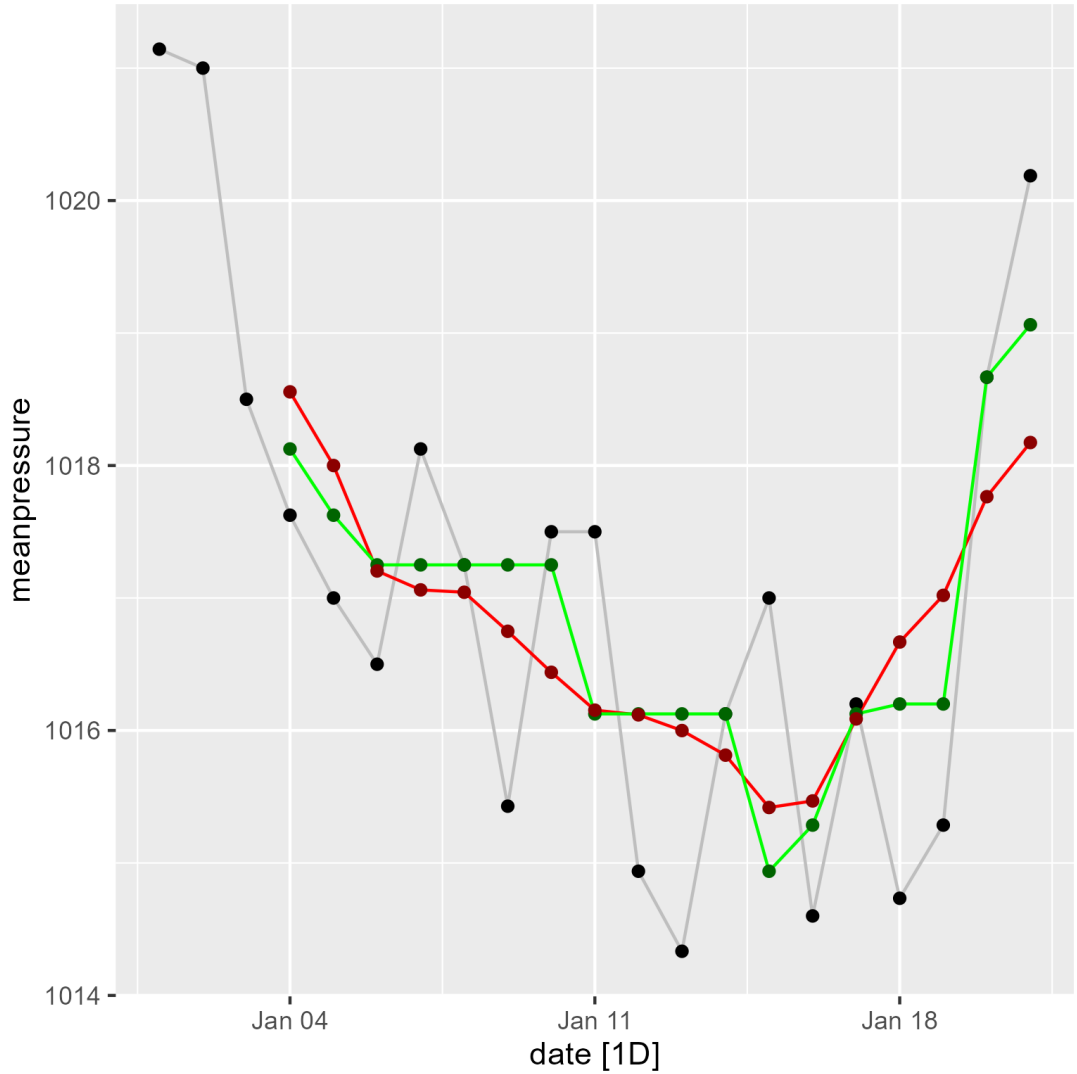
- The effect of the outlier is visible on the moving averages, which becomes significantly higher around it.
- For a higher value of m, the effect is less, it is still significantly modified compared to the others.
- For such cases with an outlier value (which might actually be a real value), a *moving median* might be a better option.
- In case of an odd m, in order to determine the median of the m values, sort them, and pick the middle one.
- In the *slider_dbl* function the moving median can be computed (instead of the moving average) if the second parameter is *median* instead of *mean*.

```
climate_ts_short <- climate_ts_short |>
 mutate(MM7 = slider::slide_dbl(meanpressure, median, .before = 3, .after = 3,
     .complete = TRUE))
```

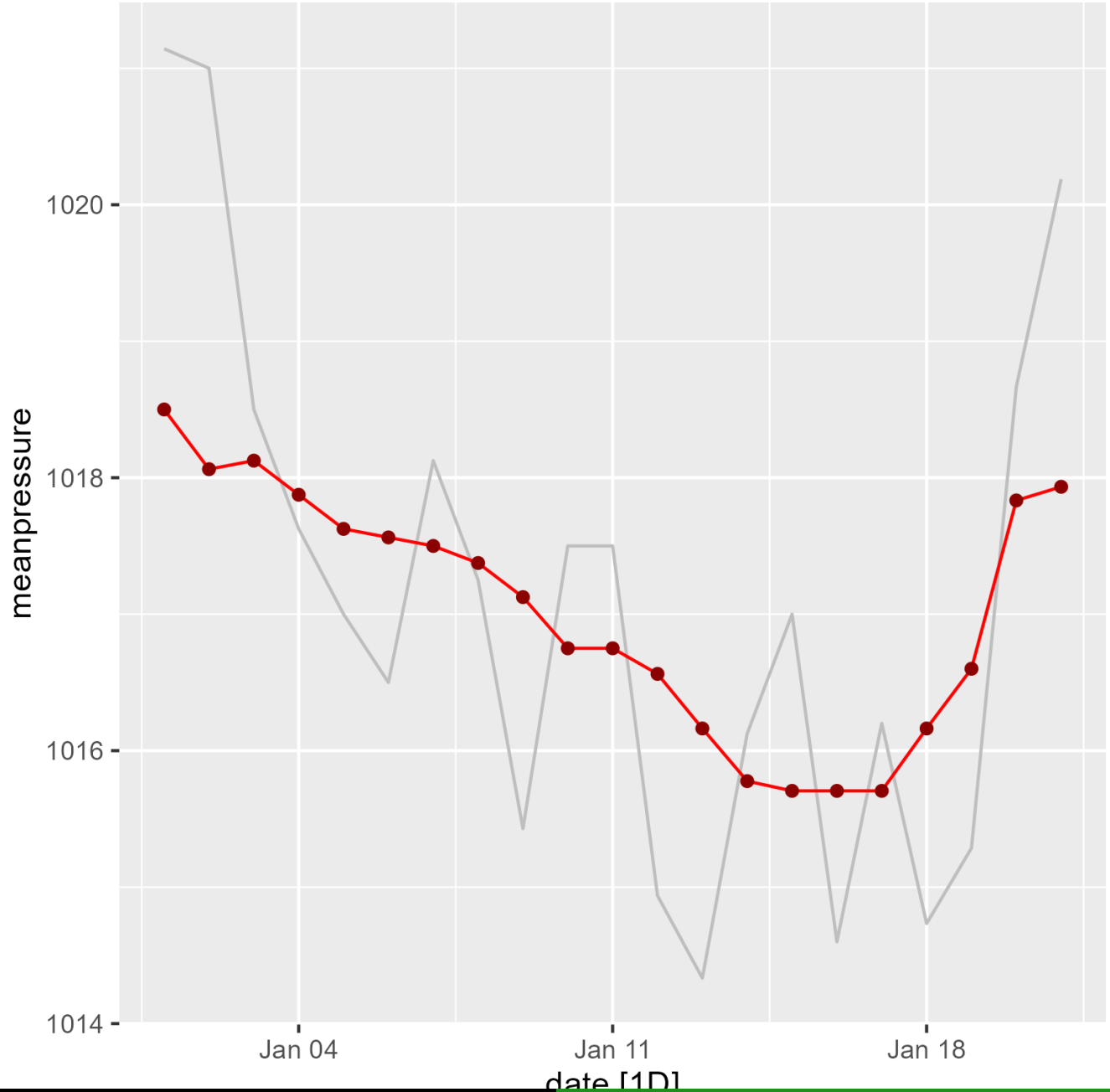
- Moving average of order 7 (red line) vs. moving median of order 7 (green line)



- The points in an odd-order moving median are actually points from the time series, while in case of even-order moving median, they are the average of the two middle values.
- 7-MA versus 7-Moving Median:



- Moving median or order 10.



History od time series decomposition models

- The classical decomposition was developed in 1920.
- In 1957 the Census II method was introduced, which was the basis for the X-11 method and its variants (X-12-ARIMA and X-13-ARIMA)
- STL method was introduced in 1983
- SEATS and TRAMO were introduced in the 1990s.

Methods used by official statistics agencies

- Official statistics agencies have developed their own decomposition methods which are used for seasonal adjustments. Most of them are based on the so-called *X-11* or *SEATS* methods (or a combination of them), which are methods suitable for quarterly and monthly data only.
- In the *seasonal* package a series of such methods are implemented.

Classical decomposition method I

- Assuming that we have a time series, with a seasonal period of m the classical additive decomposition (where we assume $y_t = S_t + T_t + R_t$ has the following steps:

  1. If m is even, compute the $2xm - MA$, otherwise compute the $m - MA$ of the series. This will be the *trend-cycle* component, $\hat{T}_t$.

  2. Compute the detrended series: $y_t - \hat{T}_t$

Classical decomposition method II

3. Estimate the seasonal component, by computing for every season the average of the detrended values for that season (for example, for quarterly data, compute the average of all Q1 values, the average of all Q2 values, etc.). Adjust the seasonal components to make sure they add up to zero. The seasonal component, $\hat{S}_t$ is the sequence of these values, replicated for each year of the data.

4. Compute the remainder, by removing the trend-cycle and seasonal components from the data: $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$

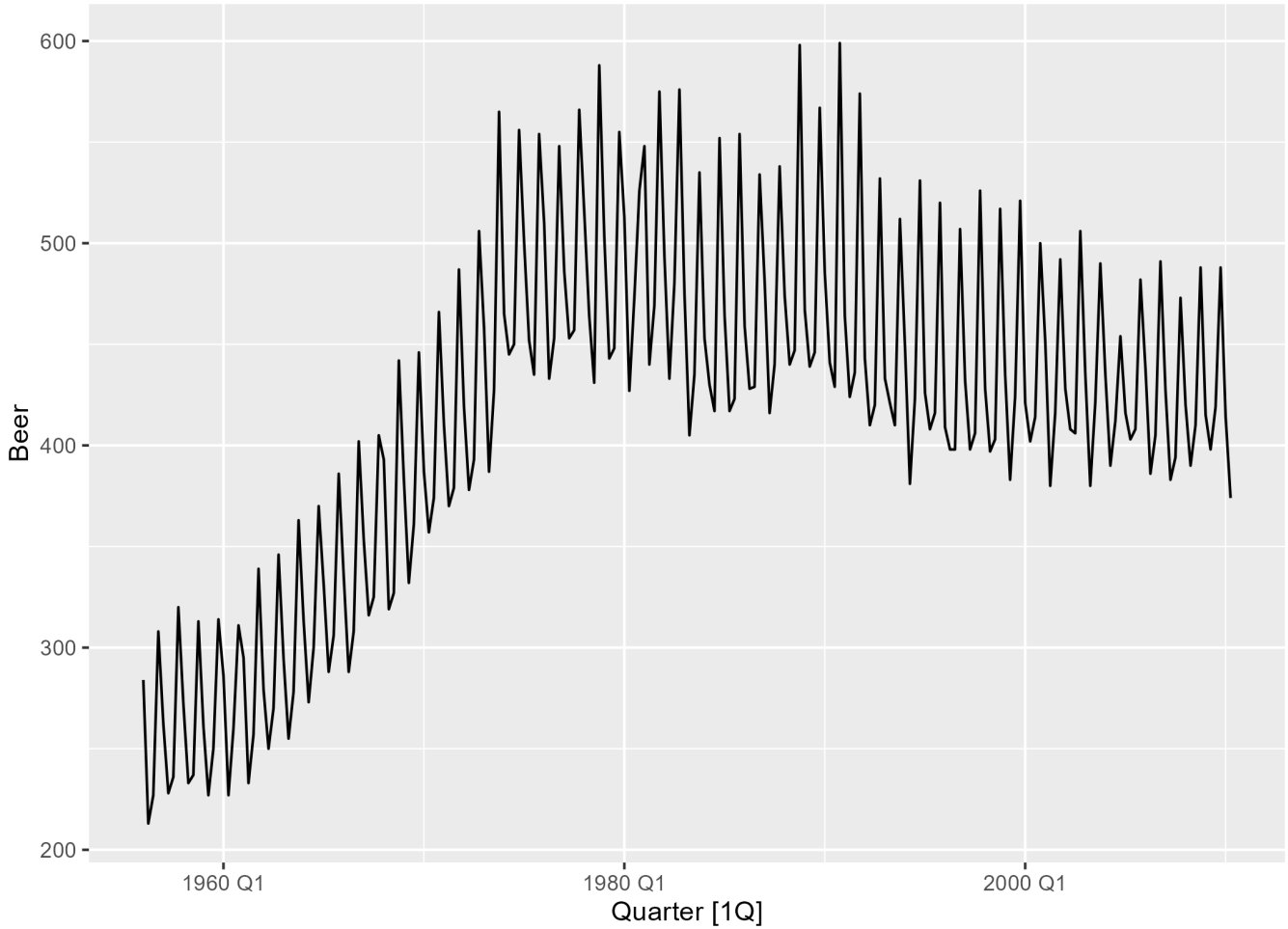
- **Obs.** Multiplicative decomposition is very similar, with the observation that we use division instead of subtraction.

- We have the classical decomposition implemented in R as well, and it takes a parameter called *type*, which can be "additive" or "multiplicative", to describe what type of decomposition to use.

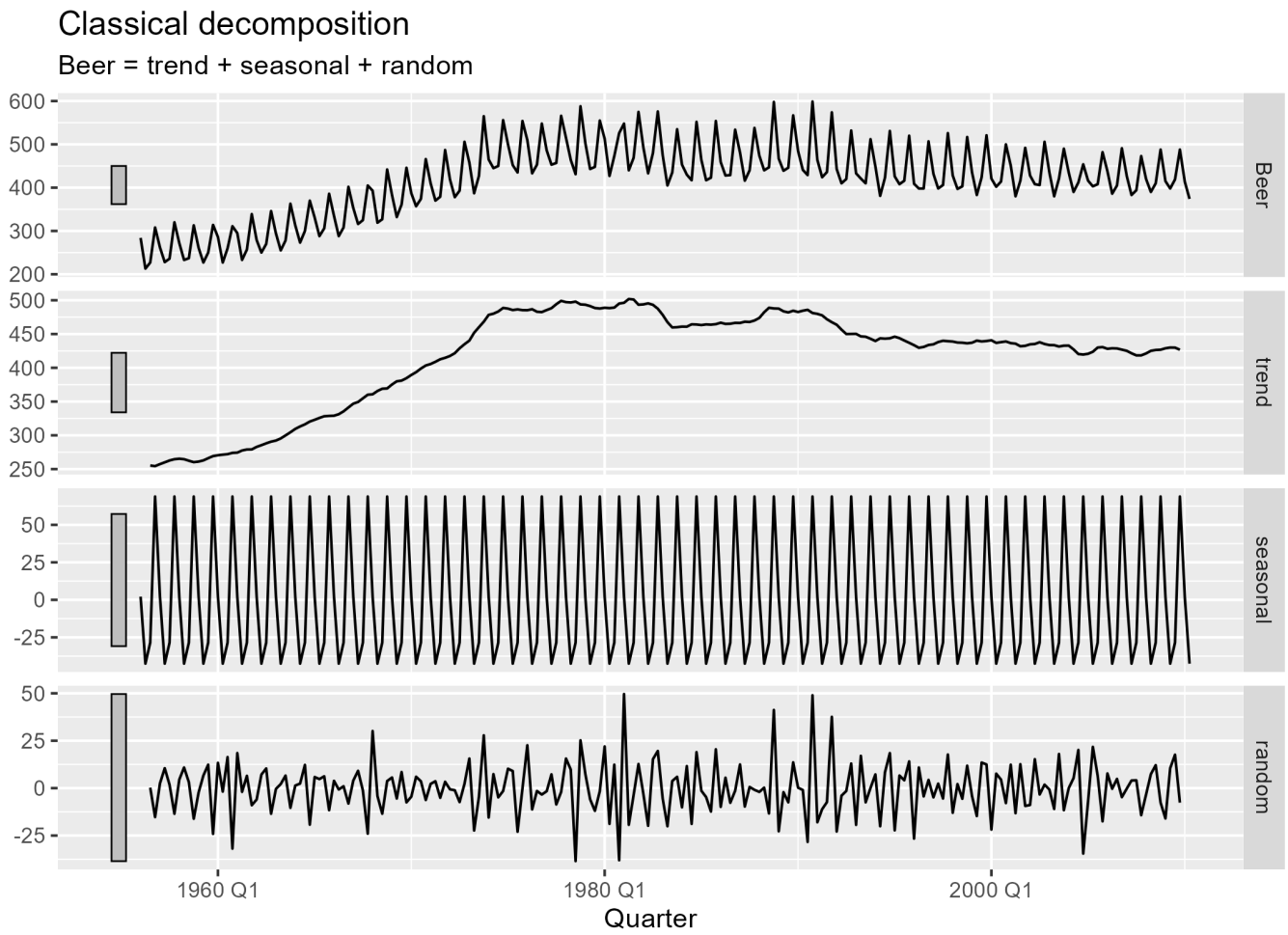
- From the components of the decomposition, we can see that the remainder is here called *random*, but it represents the same concept: what is left when you eliminate the trend and the season.

```
beer <- aus_production |> select(Beer)
beer |> autoplot()

beerDecomp <- beer |>
  model(d = classical_decomposition(Beer, type = "additive"))

beerDecomp |> components()
beerDecomp |> components() |> autoplot()

beerDecomp |> components() |>
  ACF(random) |> autoplot()
```

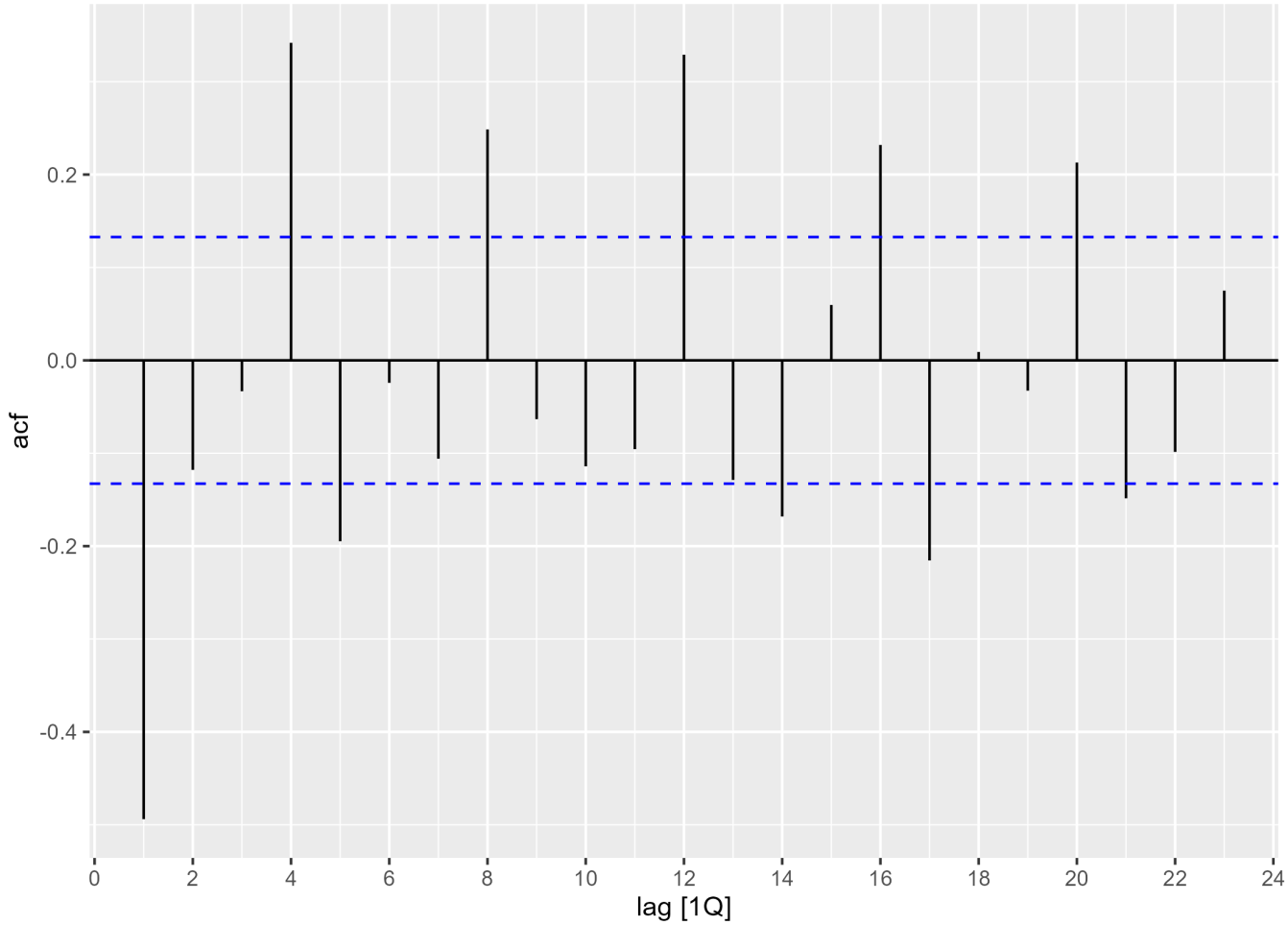
- Initial data



- Components of the classical decomposition



- ACF of the random component to see how much it resembles white noise



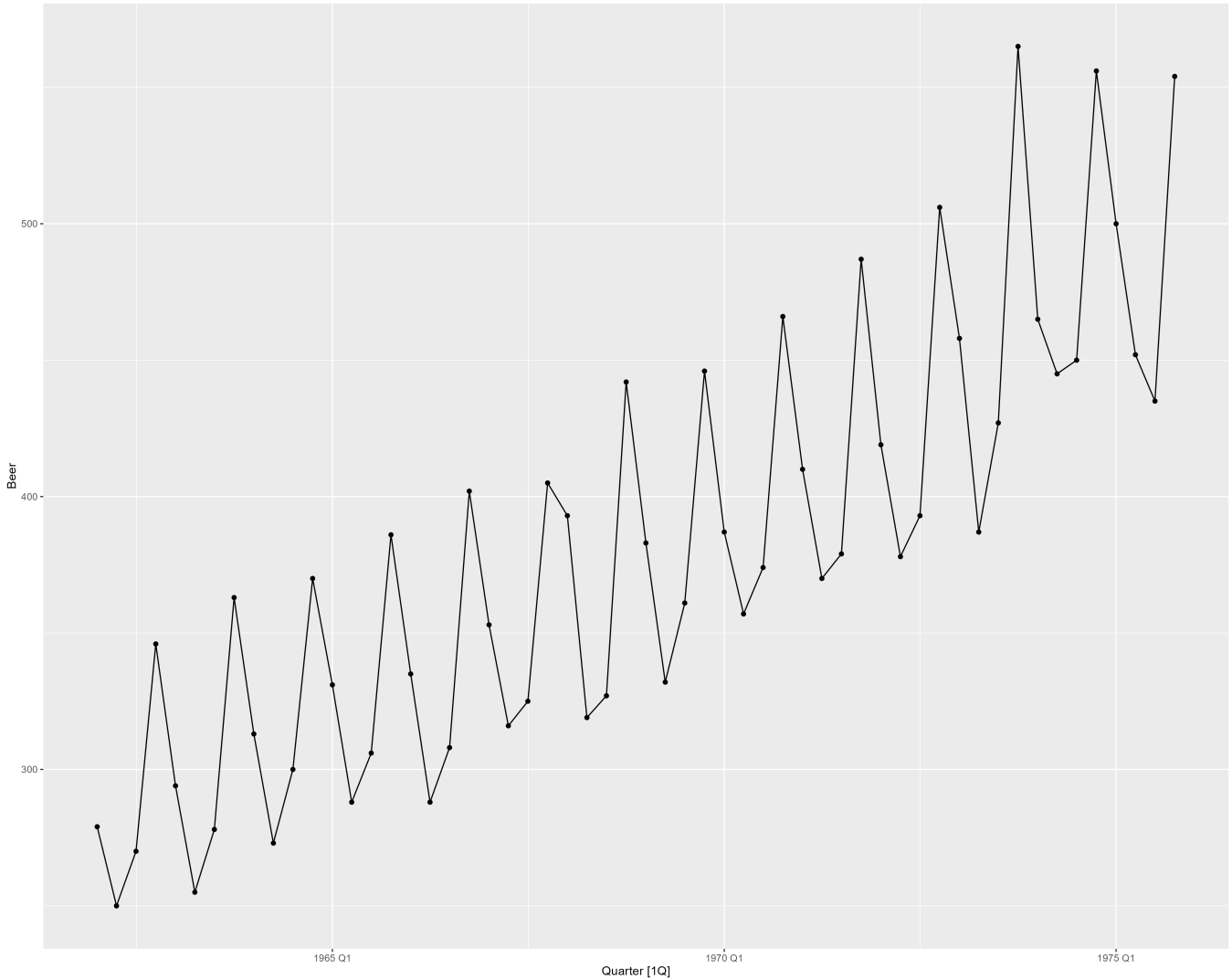
- Not at all.

- One possible reason for such a bad performance for the decoposition is that this is a long time series (218 observations, over 54 years worth of data) and the classical decomposition creates the same seasonal component for the entire period.

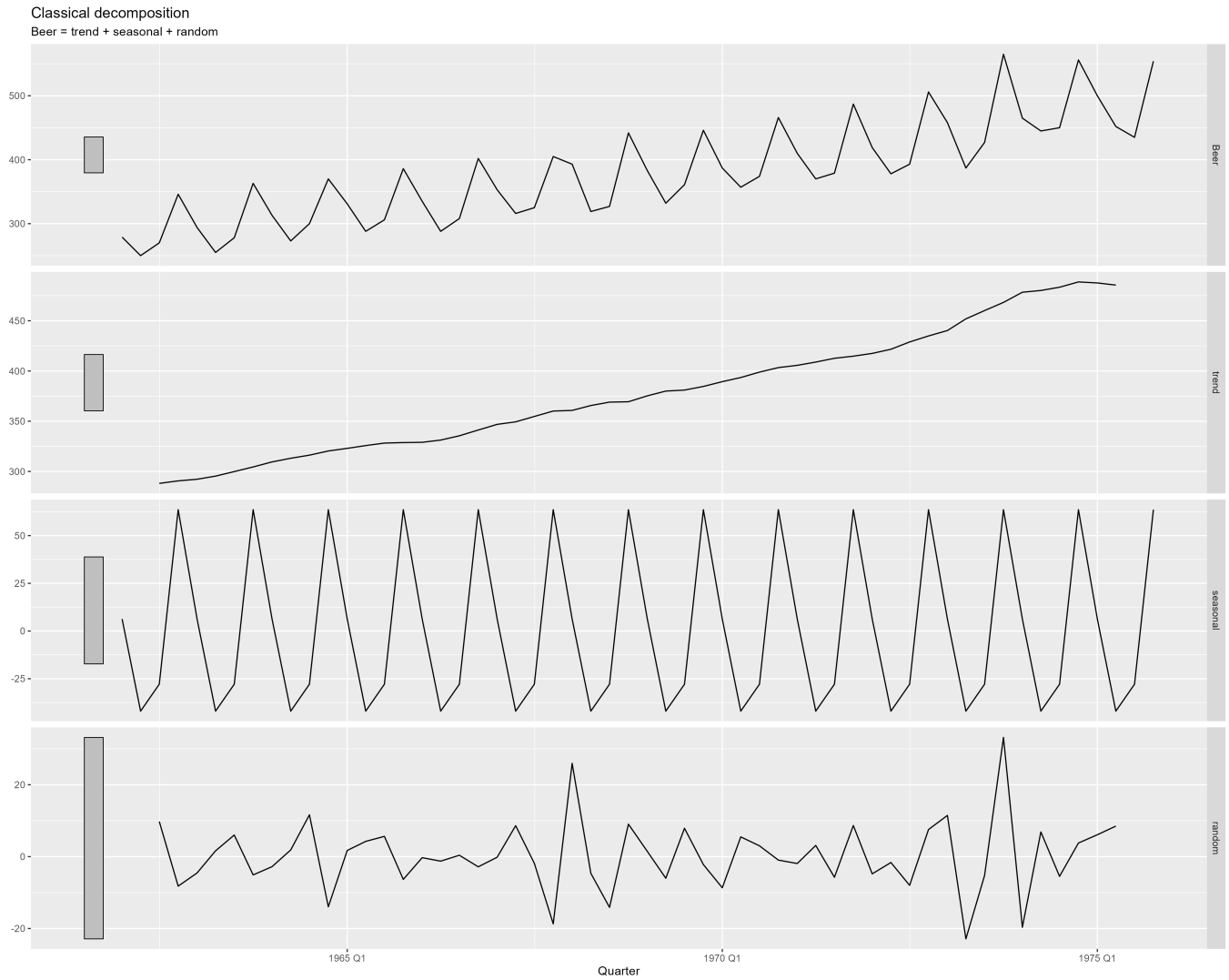
- Let us look at a shorter period, for example between 1962 and 1975.

```
beerShort <- beer |>
  filter(year(Quarter) >= 1962, year(Quarter)< 1976)

beerShort |> autoplot()
beerShortDecomp <- beerShort |>
  model(d = classical_decomposition(Beer, type = "additive"))

beerShortDecomp |> components() |> autoplot()
beerShortDecomp |> components() |>
  ACF(random) |> autoplot()
```

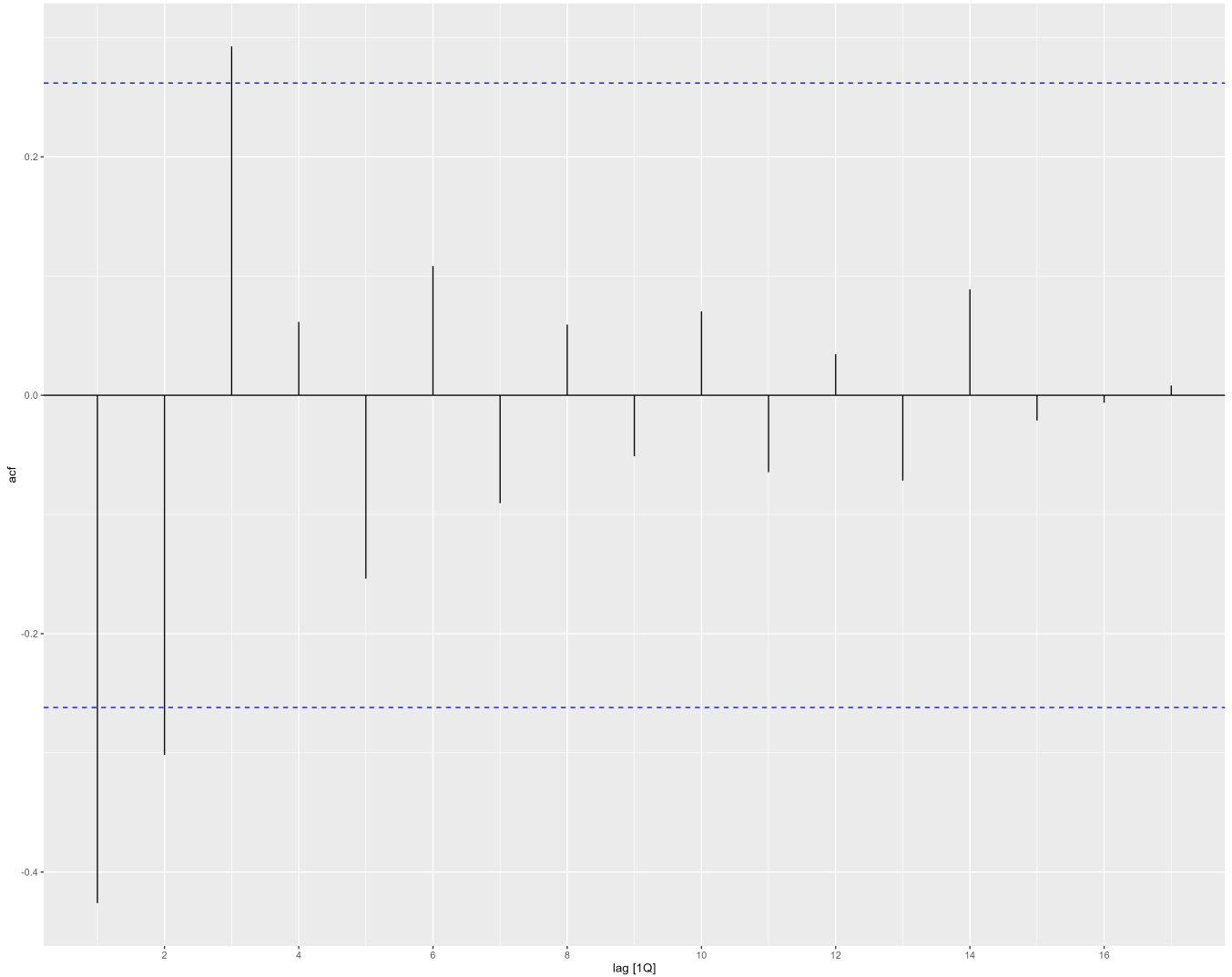
- Initial data between 1962 and 1975



- Components of the classical decomposition



- ACF of the random component

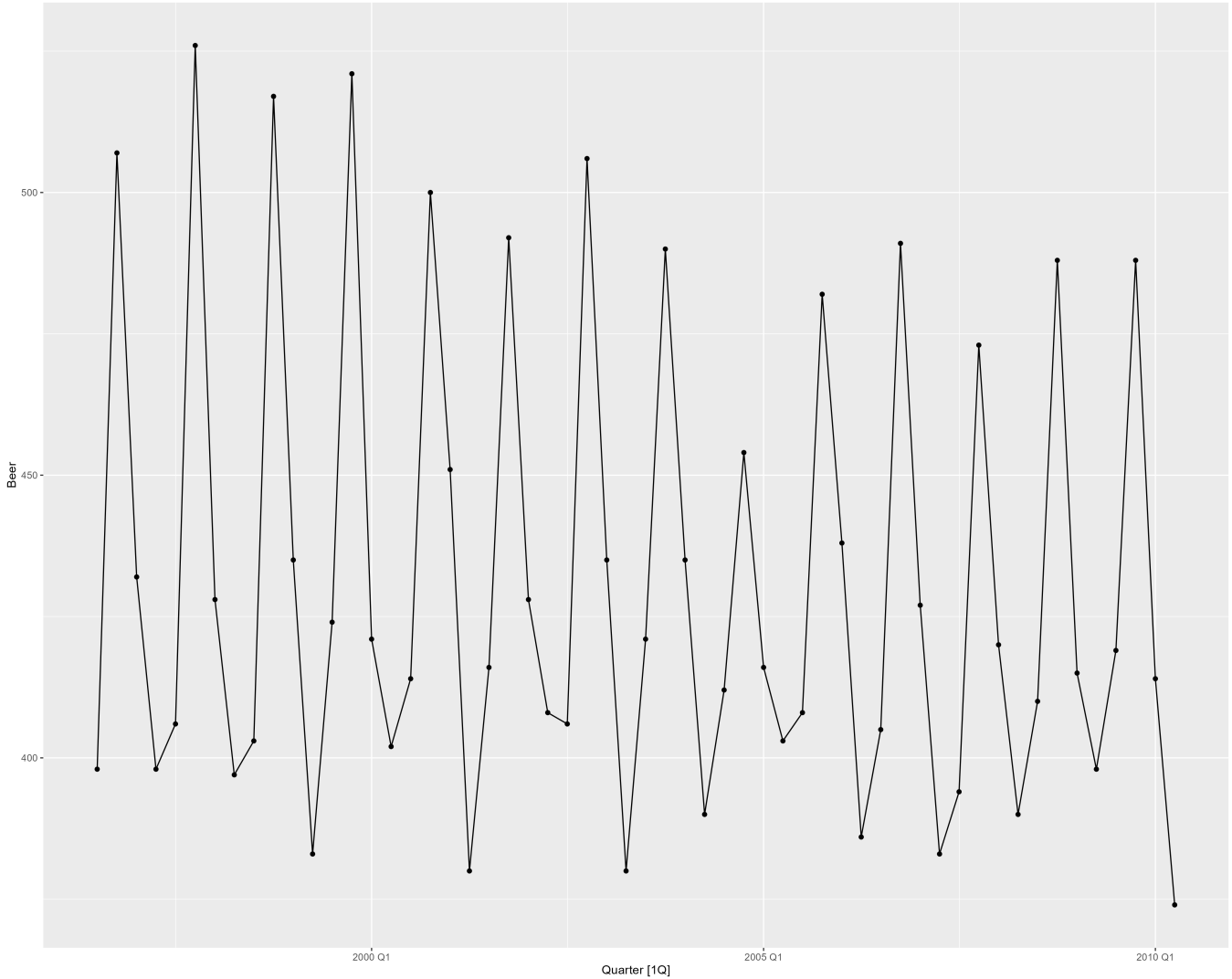


- Still not white noise, but a lot better

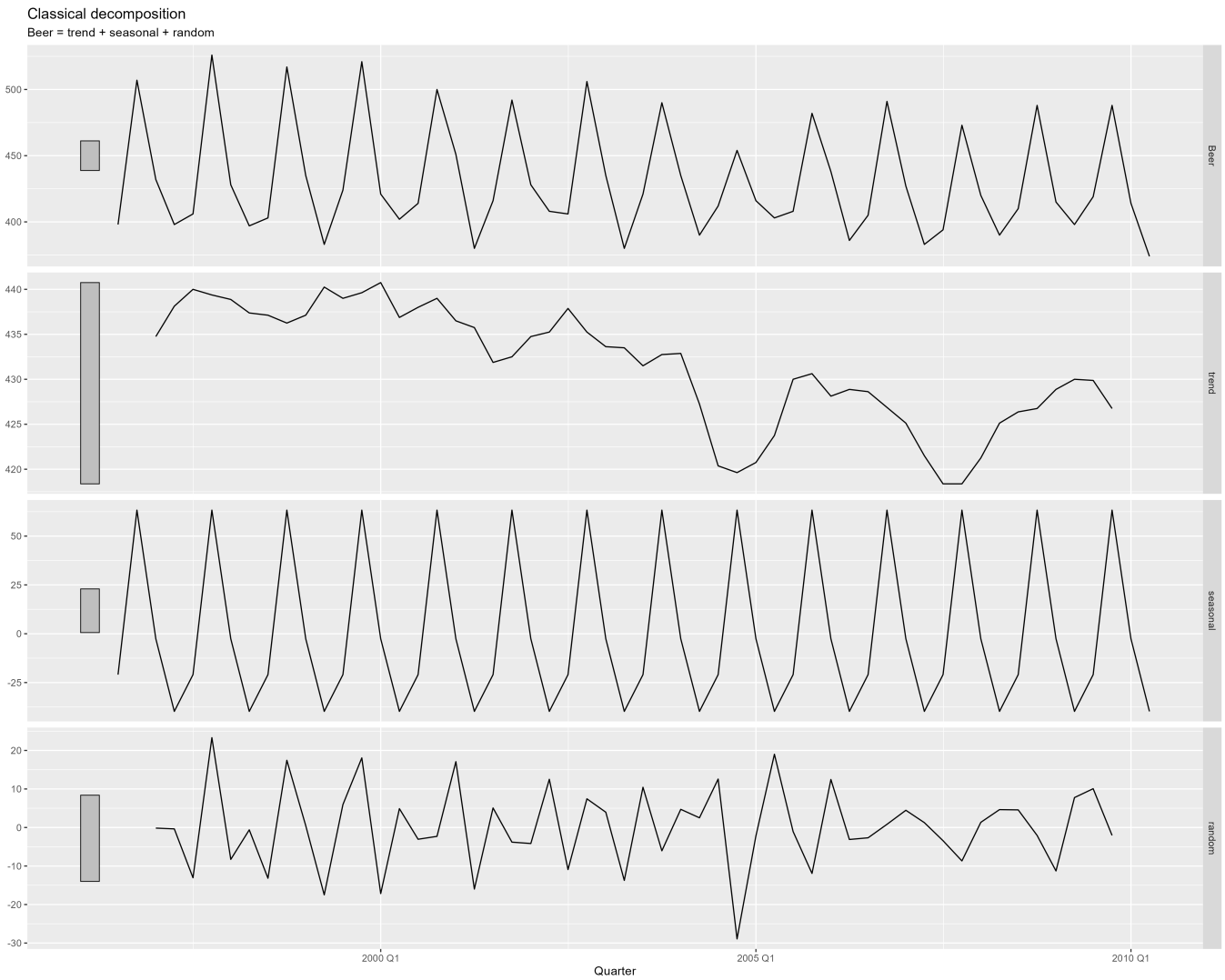
- Normally, we are more interested in recent data, so let us try the same classical decomposition on the last 56 observations as well.

```
beerRecent <- beer |> tail(n = 56)
beerRecent |> autoplot() + geom_point()

beerRecentDecomp <- beerRecent |>
  model(d = classical_decomposition(Beer, type = "additive"))

beerRecentDecomp |> components() |> autoplot()

beerRecentDecomp |> components() |>
  ACF(random) |> autoplot()
```

- Initial data for the observations starting from 1996 Q3

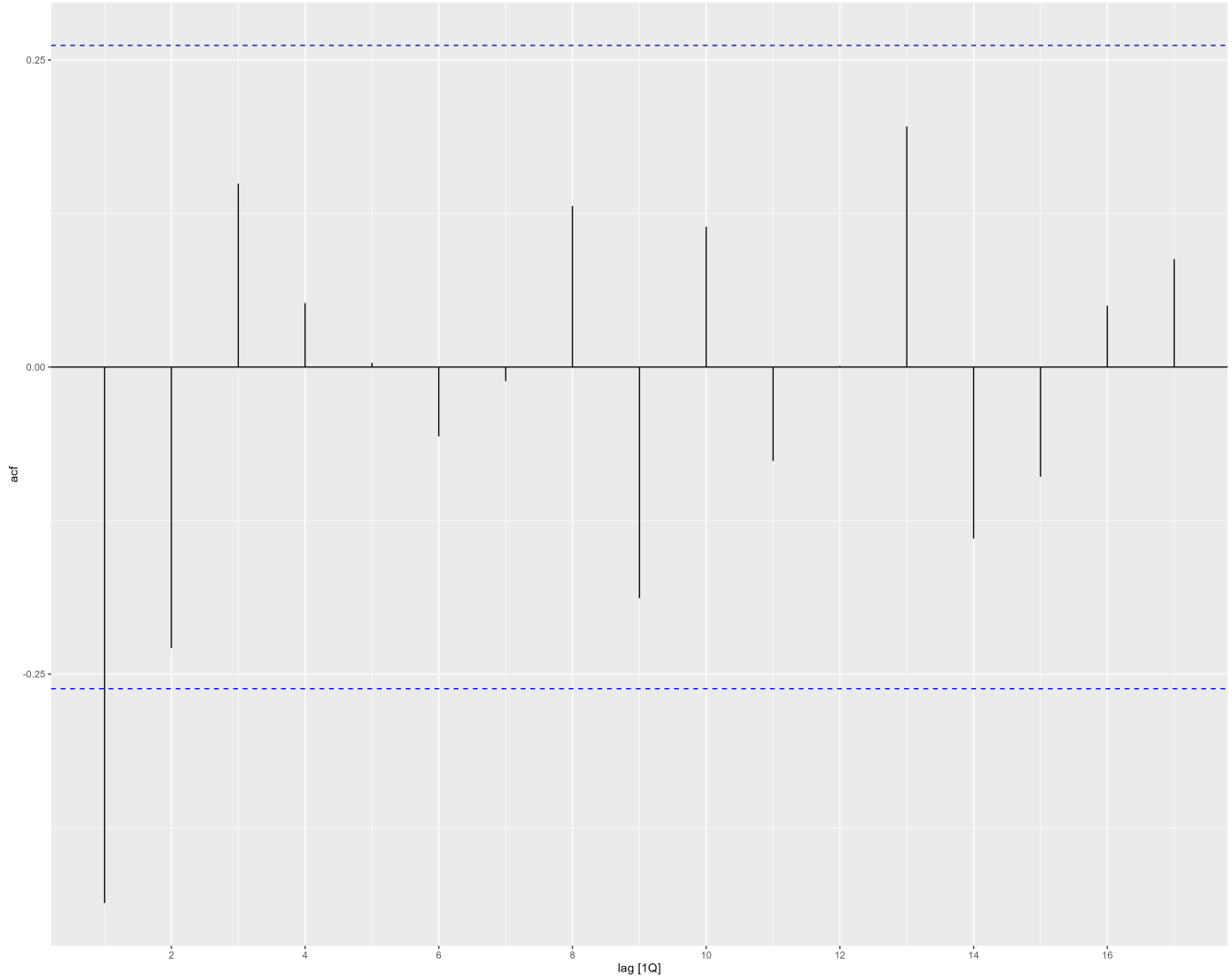


- Components of the classical decomposition



- What do you observe on this plot? Which component looks a little strange?

- ACF of the random component

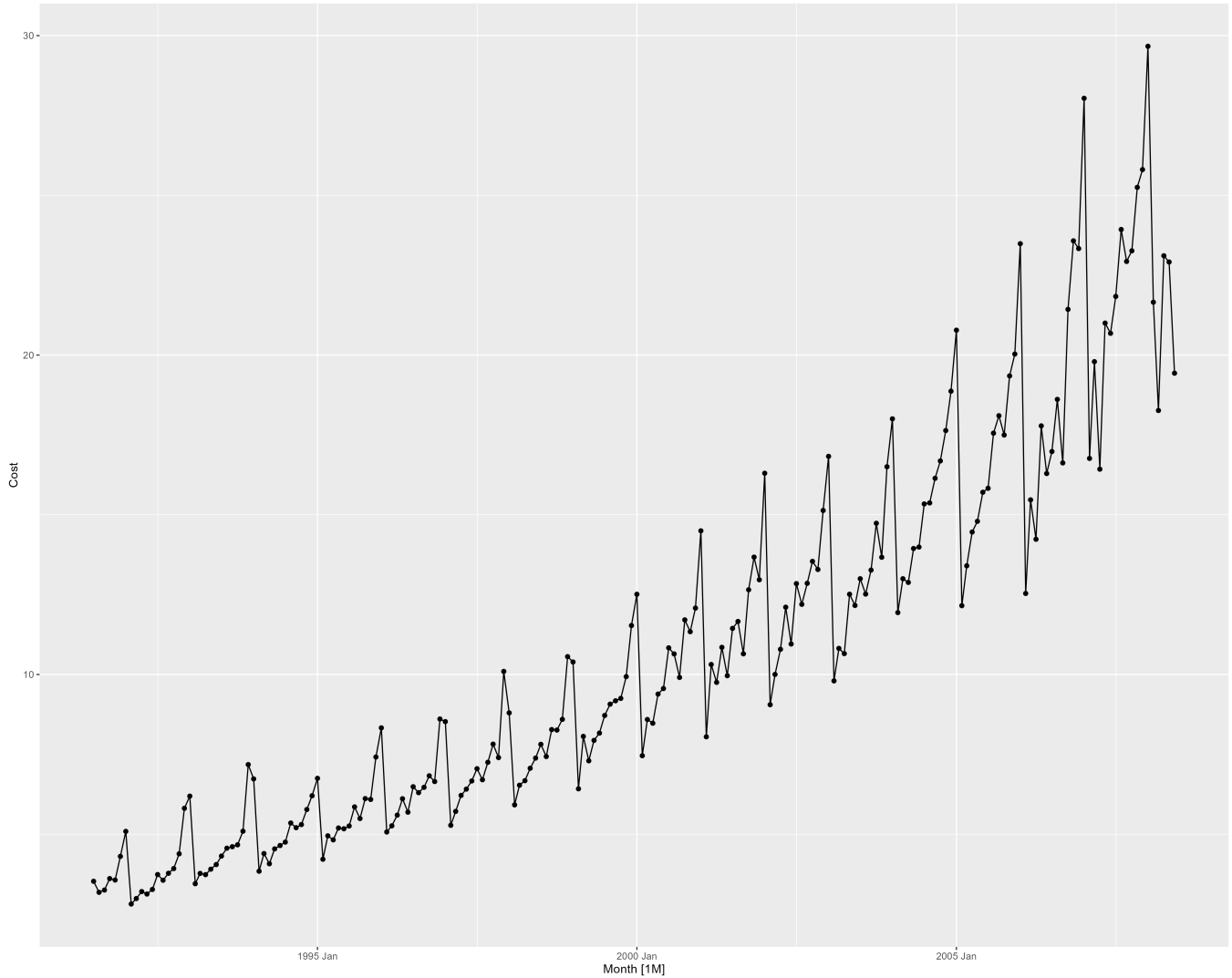


Multiplicative decomposition

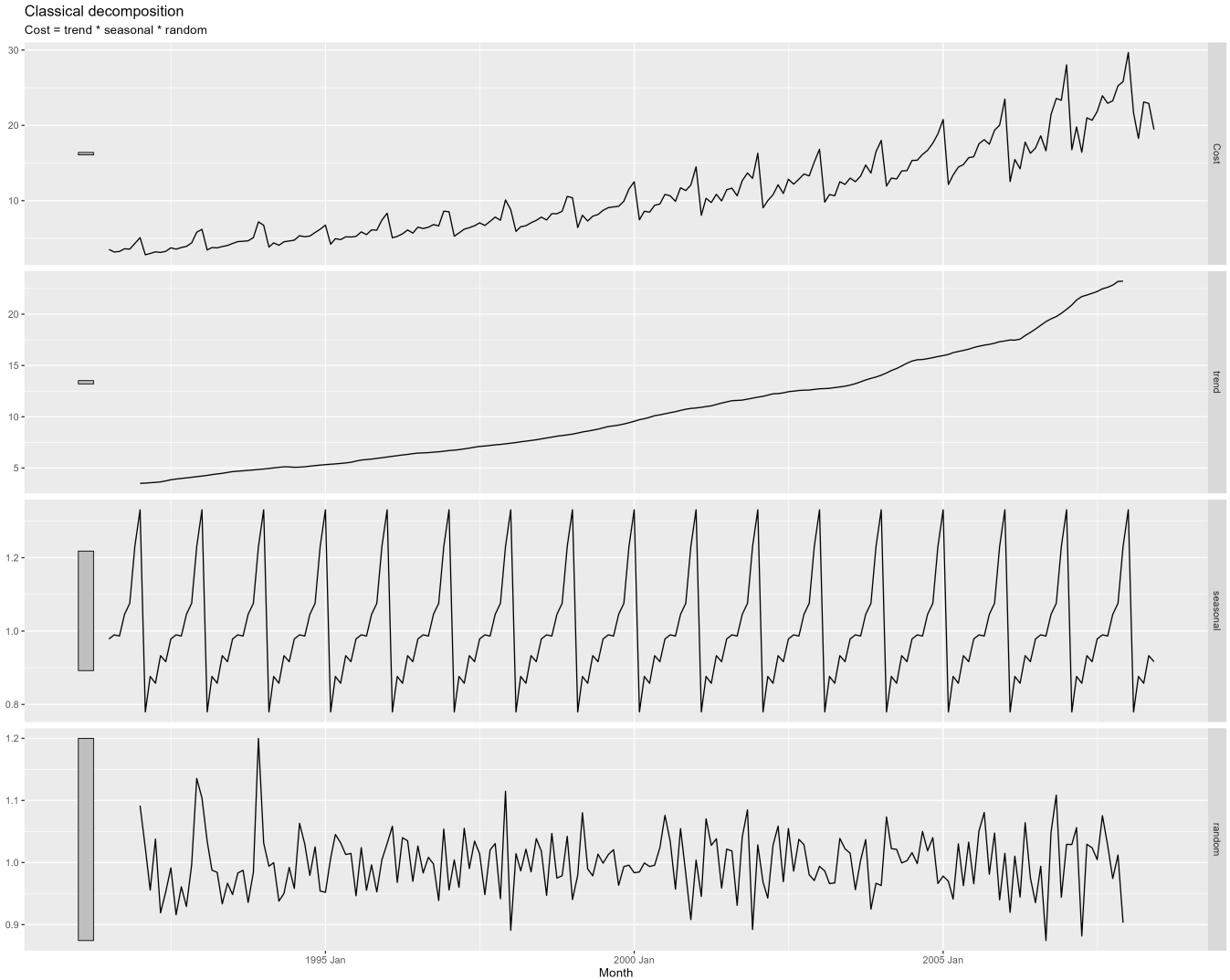
- Let us look at an example where a multiplicative decomposition is more suitable.

```
A10 <- PBS |>
  filter(ATC2 == "A10") |>
  select(Month, Concession, Type, Cost) |>
  summarize(TotalCost = sum(Cost)) |>
  mutate(Cost = TotalCost / 1000000)

A10 |> autoplot(Cost) + geom_point()
A10Decomp <- A10 |> model(d = classical_decomposition(Cost, type = "
    multiplicative"))
A10Decomp |> components()

A10Decomp |> components() |> autoplot()
A10Decomp |> components() |>
  ACF(random) |> autoplot()
```

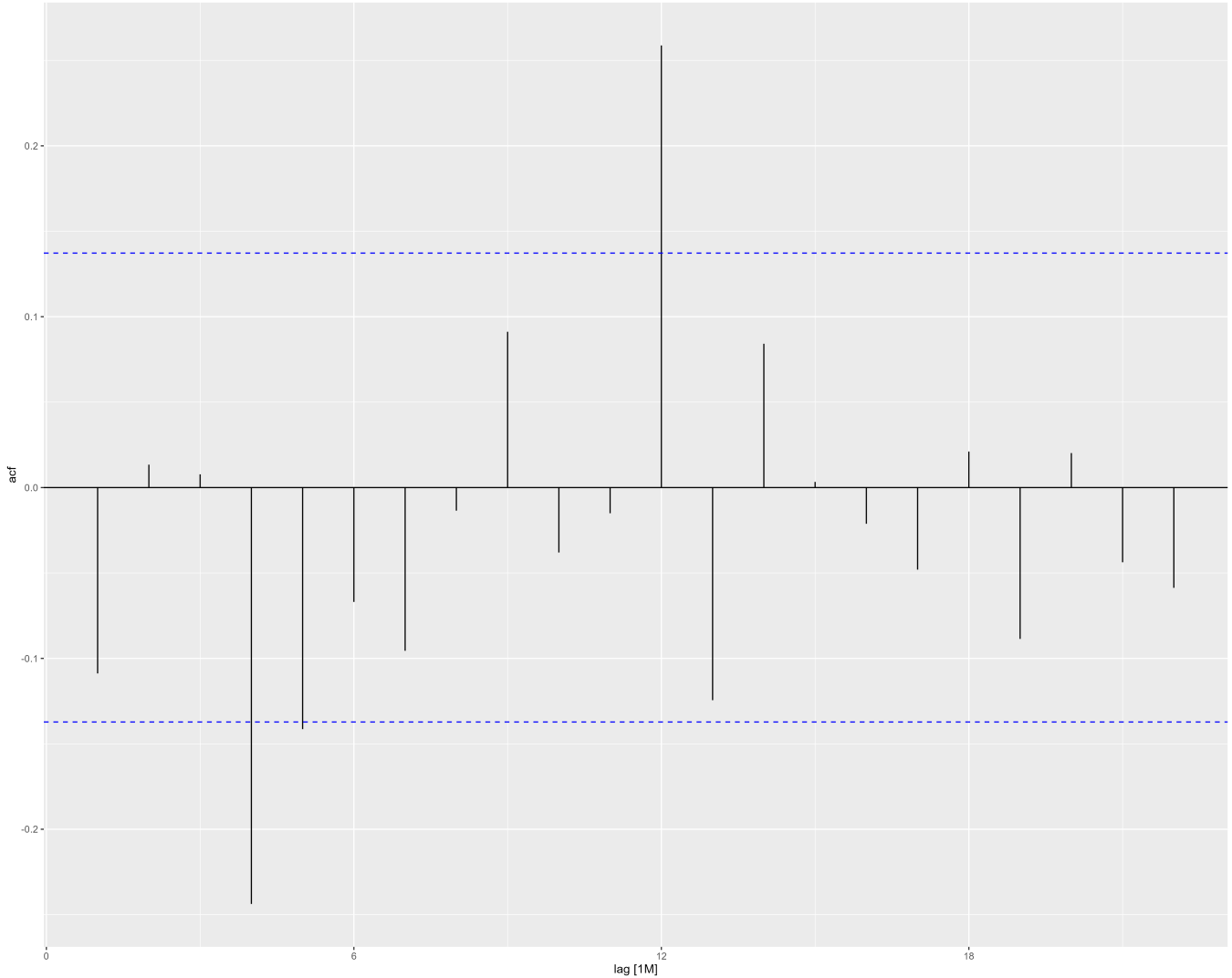
- Initial data



- Components of the classical decomposition



- ACF of the random component



- Some problems/disadvantages of the classical decomposition method:
  - You do not have trend-cycle estimates for the first and last few observations (since the moving average is not computed for these).
  - The trend-cycle tends to over-smooth rapid rises and falls in data.
  - It cannot capture if seasonal patterns change over time (it just repeats the same sequence of data)
  - Not robust for unusual data (for example: changes in monthly air passenger traffic due to a strike).

- In practice, it is not recommended to use the classical decomposition anymore, since there are better alternatives.

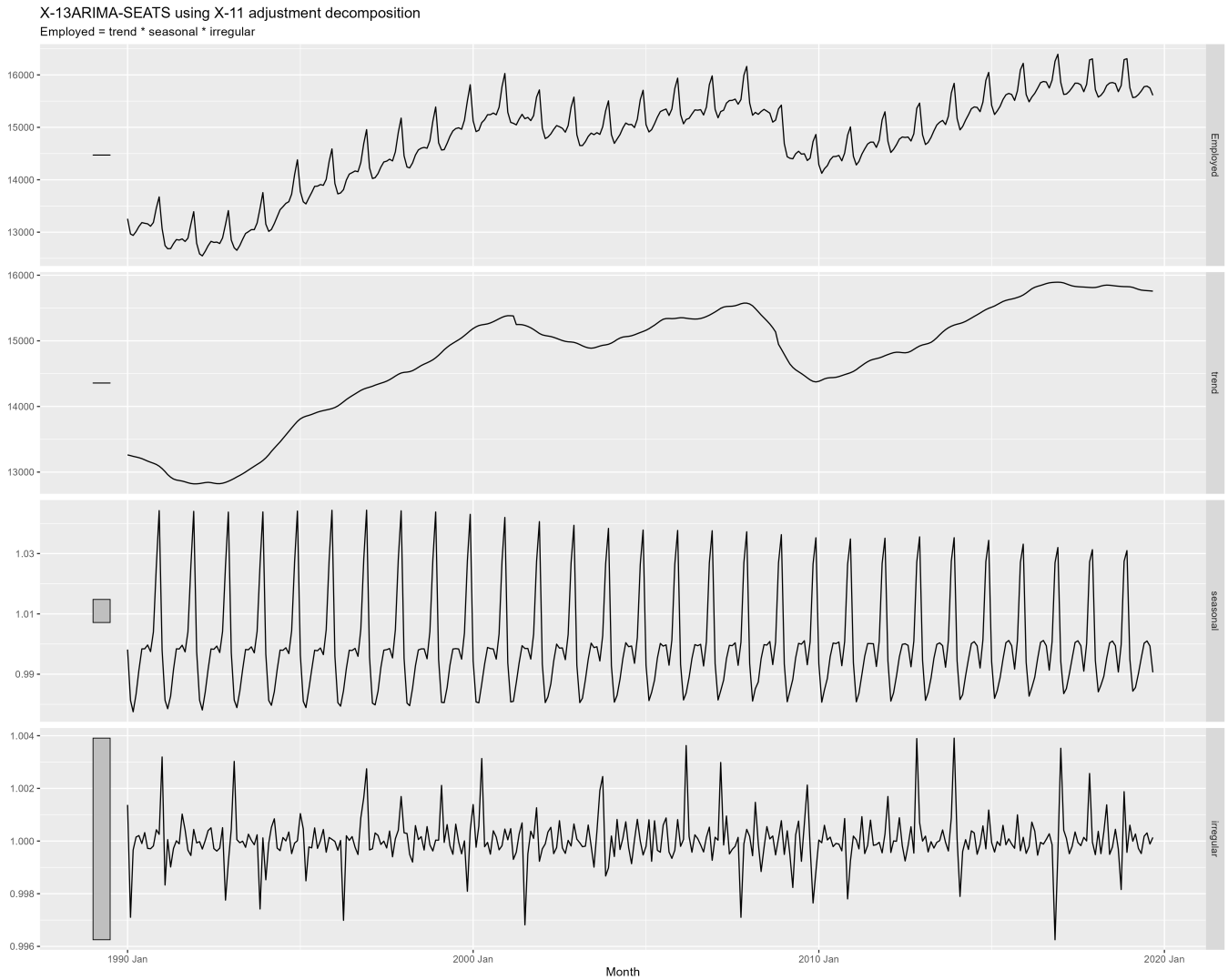
X-11 decomposition I

- Originated from the US Census Bureau and Statistics Canada.
- It is based on the classical decomposition, but has many extra steps to overcome the drawbacks of classical decomposition:
  - Trend-cycle estimates are available for all data points
  - The seasonal component is allowed to vary slowly over time
  - It can handle holiday effects and trading day variation (the fact that not all months have the same number of working days)
- It can do both multiplicative and additive decomposition
- The model is automatic and robust to outliers
- As a disadvantage, it can only work with monthly and quarterly data.

X-11 decomposition II

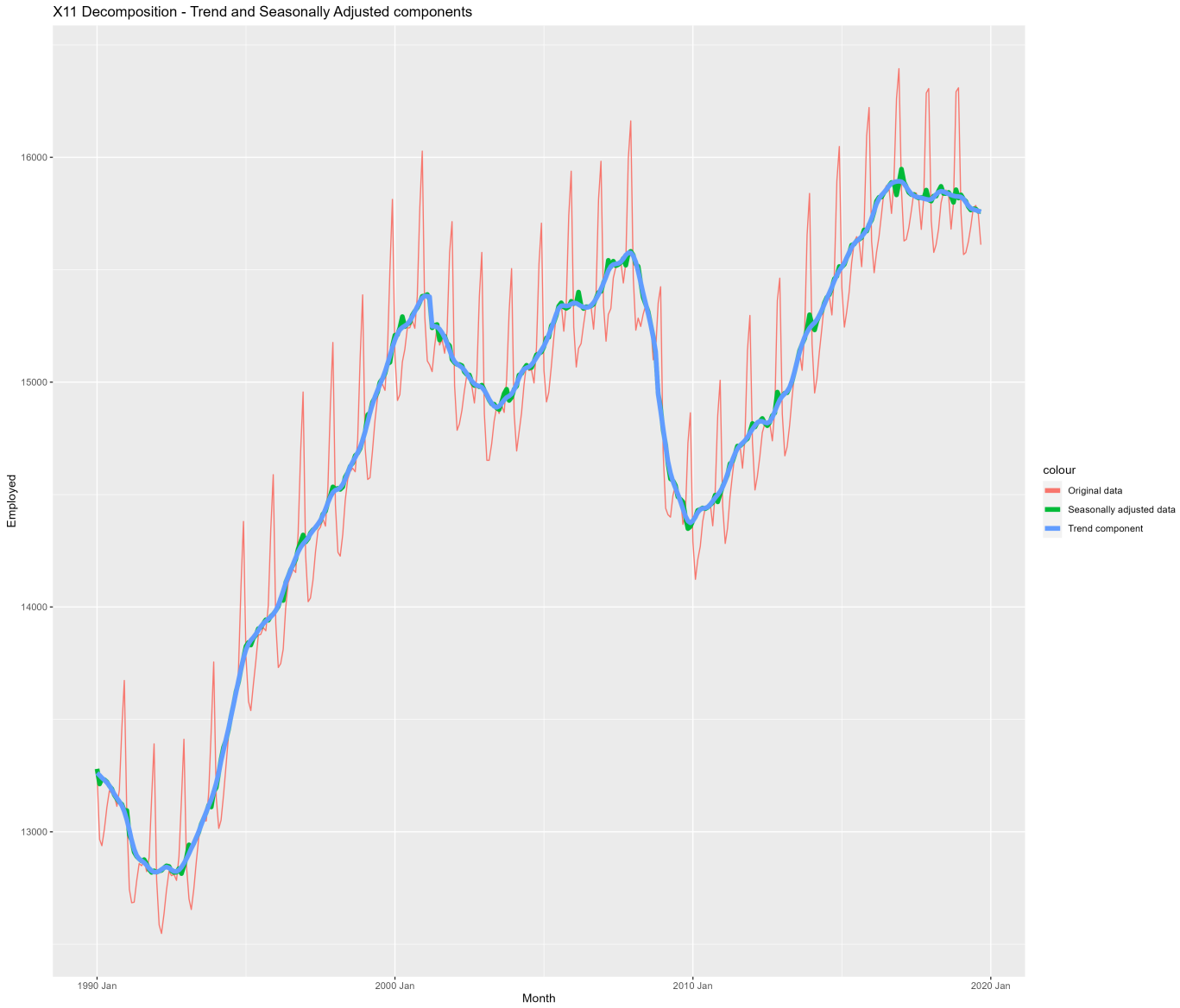
- In R, it can be found as part of the *X_13ARIMA_SEATS* group of methods (so, you create an *X_13ARIMA_SEATS* model, and then you specify as parameter that you want to use the *x11* method)

```
x_11_decomp <- us_retail_employment |>
  model(x11 = X_13ARIMA_SEATS(Employed ~ x11()))

x_11_decomp |> components() |> autoplot()

x_11_decomp |> components() |> tail(n = 10)
```



- What do you notice on the plot?

- If we plot the trend and the seasonally adjusted data (which is trend + remainder) together, we can see how well the trend covers the data


X11 Decomposition - Trend and Seasonally Adjusted components
16000
15000
14000
13000
Employed
1990 Jan
2000 Jan
2010 Jan
2020 Jan
Month
colour
Original data
Seasonally adjusted data
Trend component

- Some details about the implementation of X11 can be found on the webpage of the *Australian Bureau of Statistics*: `https://www.abs.gov.au/websitedbs/d3310114.nsf/4a256353001af3ed4b2562bb00121564/c890aa8e65957397ca256ce10018c9d8!OpenDocument`

- According to the above document, the X11 algorithm is the following, assuming monthly data and multiplicative decompositions:

  1. Use a 2x12-MA to create an initial estimate of the trend component. 6 values at both ends will not have trend values.
  2. Remove the trend from the series (by dividing by it), leaving only the seasonal and irregular/random/remainder components.

3. Build a preliminary estimate of the seasonal component, by applying a 3x3-MA on the detrended season for each month separately. Missing values at the ends are replaced by repeating the previous year's pattern.
4. Compute the seasonally adjusted time series, by dividing the original one by the seasonal component.
5. Re-estimate the trend, by applying a 9, 13 or 23 term Henderson moving average to the seasonally adjusted values. Use asymetric filters at the end to have an estimate for every time step.
6. Divide the original time series by this new estimate of trend and re-estimate the season again (repeat step 3).
7. Final estimate of seasonally adjusted data is computed by dividing the original time series by this new seasonal component.
8. Compute the final estimate of the trend component by applying a 9, 13 or 23 term Henderson moving average.

9. Compute the final estimate of the irregular component, by diving the time series by the trend and the seasonal component.

- The Henderson moving average is a weighted, symmetric moving average, which works with special, pre-defined weights. For example, for the 13 term Henderson moving average we have the following weights:
  - -0.02, -0.03, 0, 0.07, 0.15, 0.21, 0.24, 0.21, 0.15, 0.07, 0.00, -0.03, -0.02

- **Obs:** In the R implementation, you can specify the moving averages for the trend and the seasonal component estimation.
  - For trend, use the *trendma* parameter, whose value needs to be a positive odd integer.
  - For season use the *seasonalma* parameter, whose value nieeds to be "s3x1", "s3x3", "s3x5", "s3x9", "s3x15", "stable", "msr", or "default"
- What do you think is the difference between using "s3x1" and "s3x15" for the seasonal component?

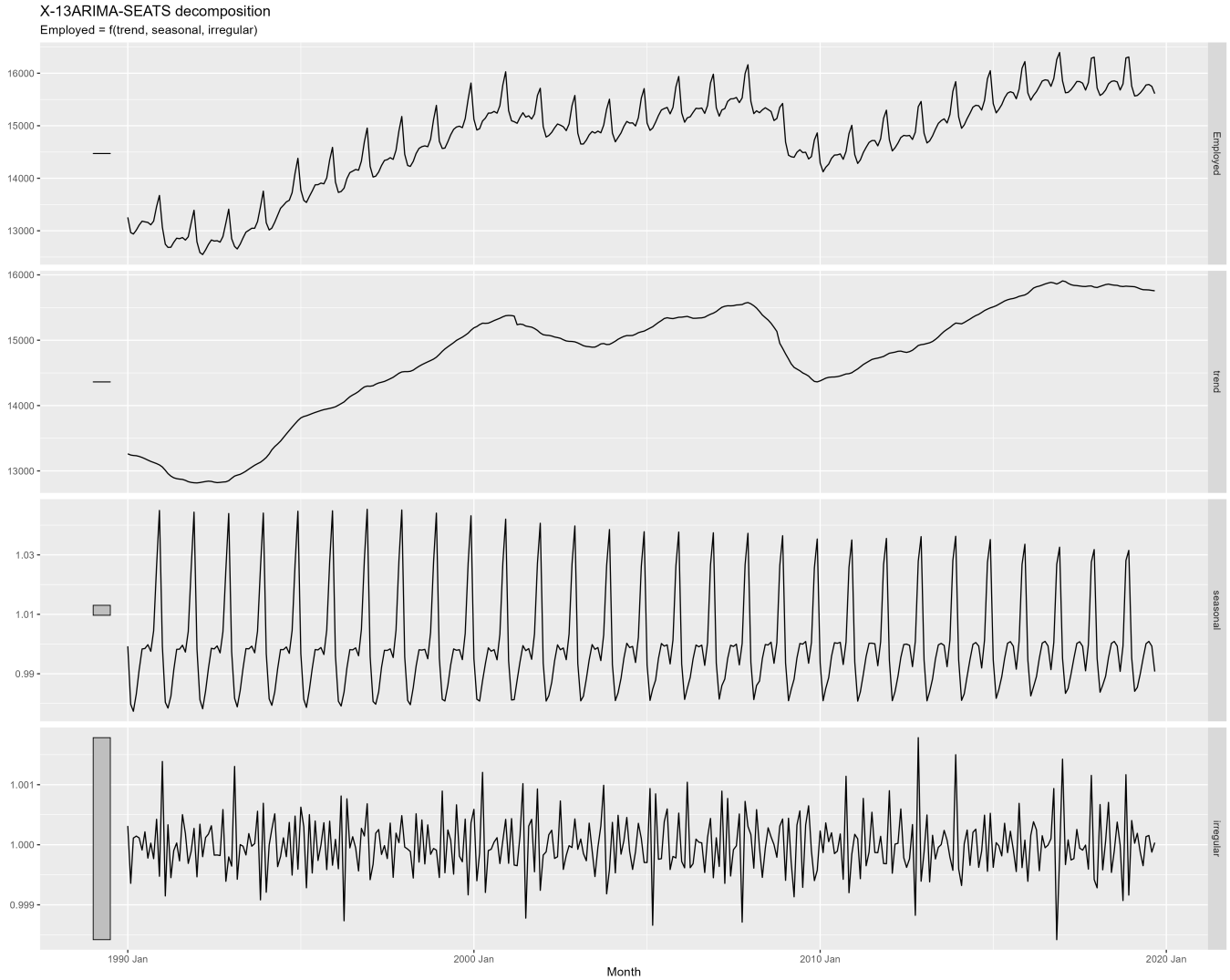
- *X-11-ARIMA*, *X-12-ARIMA* and *X-13-ARIMA* are extensions of X11.

- The ARIMA part in their name refers to the fact that the ARIMA (forecasting) method is used to extend the time series and apply the X11 algorithm on the extended one (thus, eliminating the problem of having missing values at the end of the series for the trend estimation).

SEATS method

- Stands for "Seasonal Extraction in ARIMA Time Series"

```
seats_dcmp = us_retail_employment |>
  model(seats=X_13ARIMA_SEATS(Employed ~ seats())) |>
  components()
autoplot(seats_dcmp)
```



- Based on this plot, do you think SEATS is doing additive or multiplicative decomposition?

- SEATS allows estimates at the endpoints and it allows changing seasonality, but, similar to X11, it only works for quarterly and monthly data.

STL decomposition I

- The name stands for "Seasonal and Trend decomposition using Loess" and it was developed in 1990.
- It uses a locally fitted regression model called Loess (Locally Estimated Scatterplot Smoothing), which is a generalization of moving averages and polinomial regression.
- It is an iterative process, first the series is detrended, then the seasonal component is estimated. After that the seasonal component is subtracted from the original data and the trend is estimated again, and so on. The Loess is used both for estimating the seasonal component and the trend.
- The remainder is what we have after the trend and the seasons were removed.

STL decomposition II

- Advantages of STL:
  - Flexible: it can handle a wide range of time series, including ones where the seasonal pattern changes over time (and the rate of change can be controlled by the user). It can also handle any type of seasonality (weekly, monthly, quarterly, etc.)
  - Improved accuracy: it can provide a smoother estimate of the trend than the classical decomposition. Smoothness of the trend can also be controlled by the user.
  - Automatic parameter selection: the method automatically selects the best parameters
  - Customizability: on the other hand, through different parameters, it offers the possibility to fine-tune the decomposition according to the characteristics of the data.
  - Robust: user can specify that they want a robust decomposition, and then occasional unusual observations will not affect the trend and seasonal components (only the remainder will be affected).

STL decomposition III

- Disadvantages of STL:
  - Can only do additive decomposition (for other types of time series transformations are needed).

STL decomposition IV

- In R, there are two *specials* that can be set when using the STL decomposition: one for the trend and one for the seasonal component.
- For the trend the trend-cycle window size can be set, which is the number of consecutive observations to be used for estimating the trend. It needs to be an odd number and it controls how fast the trend can change. Larger values lead to smother trends. Default is 21.
- For the seasonal component the window size can be set, which is the number of consecutive years to be used in estimating each value in the seasonal component. It needs to be an odd number but it can be set to *periodic* which means we want identical seasons over the years. Default value is 13.

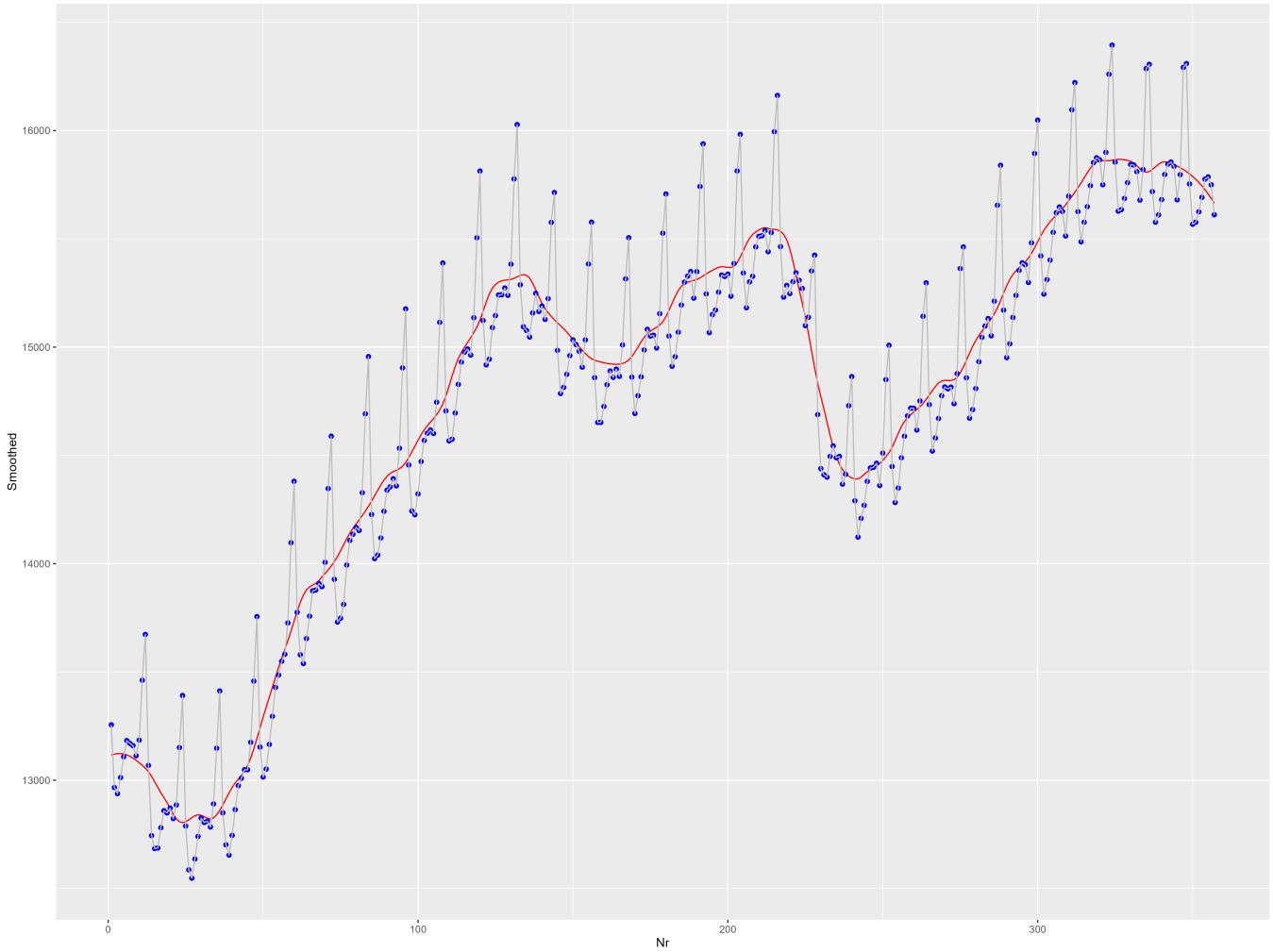
What is LOESS?

- LOESS is a regression method, which fits a curve to a set of points, but this curve is built by fitting several models on subsets of the data (for every point, only points in its neighbourhood are considered).

- In R it is implemented in the *stats* package with the *loess* function.

```
us_retail_employment_index <- us_retail_employment |>
  mutate(Nr = seq(from = 1, to = 357, by= 1))
us_retail_employment_index |> view()
us_retail_employment_index |> autoplot(Employed)


loess_model <- loess( Employed ~ Nr, data = us_retail_employment_index, span =
     0.1)
loess_model
us_retail_employment_index <- us_retail_employment_index |>
  mutate(Smoothed = predict(loess_model))
us_retail_employment_index

us_retail_employment_index |> ggplot(mapping = aes(x = Nr)) +
  geom_line(mapping = aes(y = Smoothed), color = "red") +
  geom_point(mapping = aes(y = Employed), color = "blue") +
  geom_line(mapping = aes(y = Employed), color = "grey")
```

- Example of the LOESS curve fit on the Employment data.

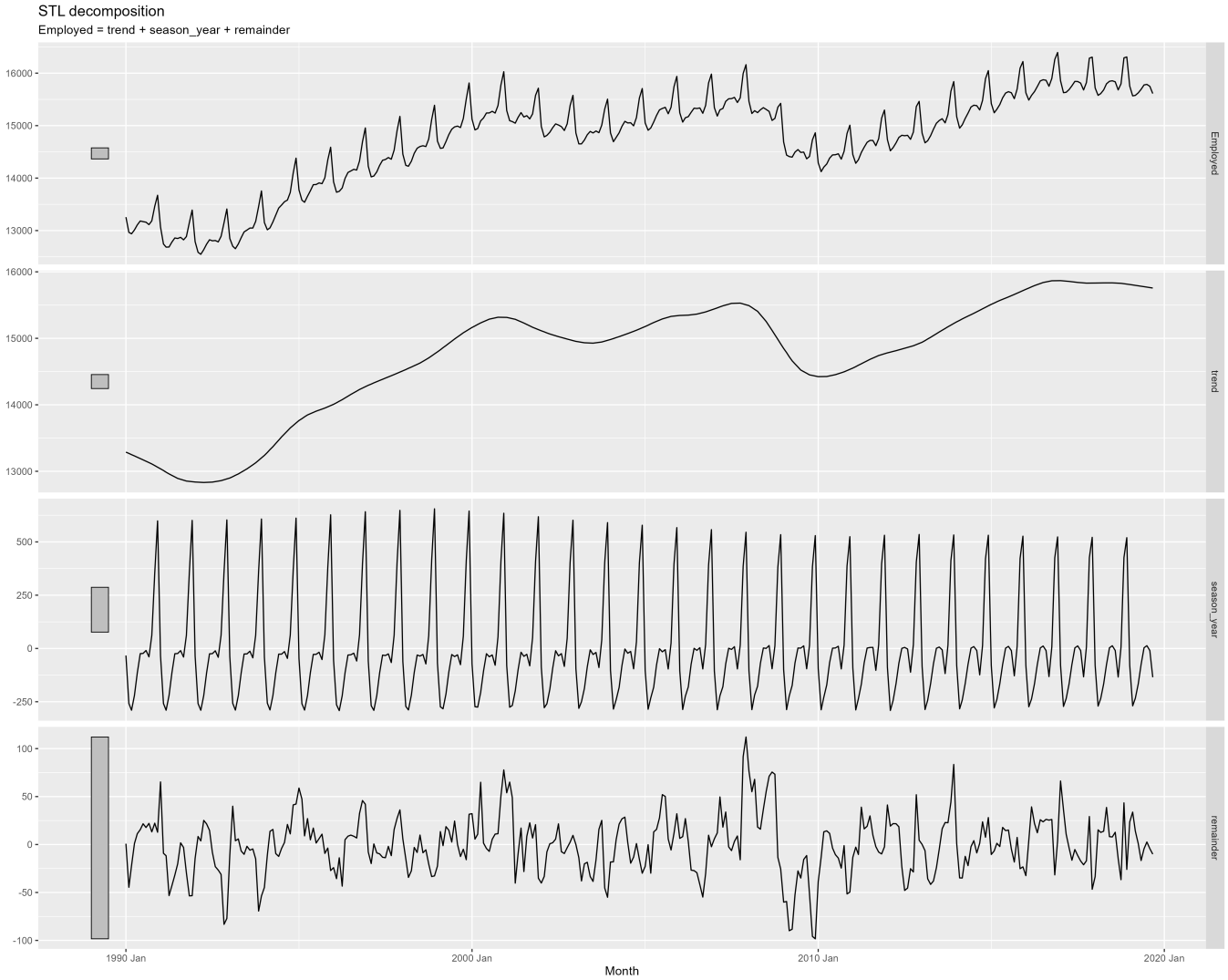


- Let us see a few examples of the STL decomposition for the US employment data with different parameter settings.

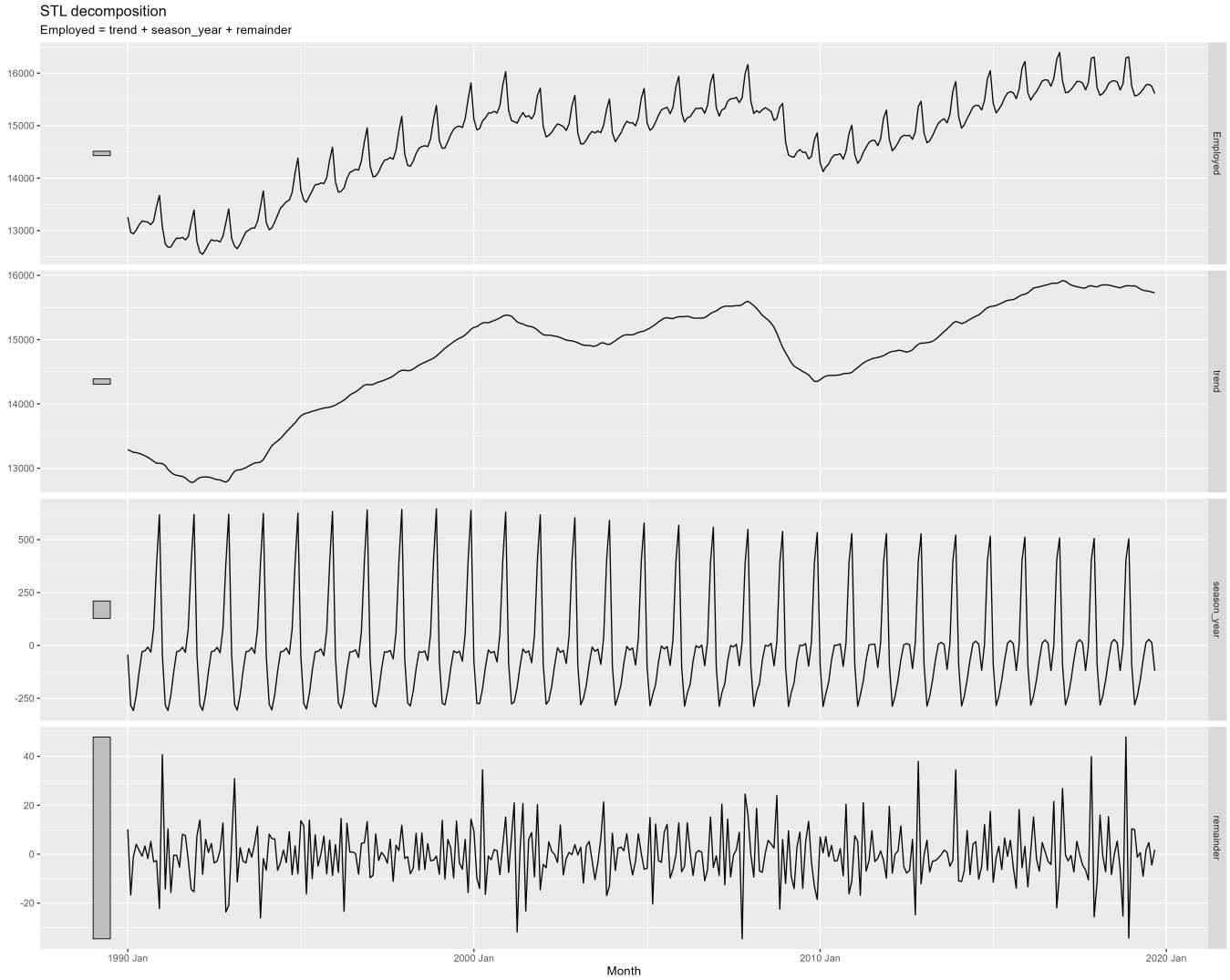
```
stl_dcm <- us_retail_employment |>
  model(Stl = STL(Employed)) |>
  components()
stl_dcm |> autoplot()

stl_dcm2 <- us_retail_employment |>
  model(Stl = STL(Employed ~ trend(window = 5))) |>
  components()
stl_dcm2 |> autoplot()

stl_dcm3 <- us_retail_employment |>
  model(Stl = STL(Employed ~ trend(window = 51))) |>
  components()
stl_dcm3 |> autoplot()

stl_dcm4 <- us_retail_employment |>
  model(Stl = STL(Employed ~ trend(window = 501))) |>
  components()
stl_dcm4 |> autoplot()

stl_dcm5 <- us_retail_employment |>
  model(Stl = STL(Employed ~ season(window = 5))) |>
  components()
stl_dcm5 |> autoplot()
```

```
stl_dcm6 <- us_retail_employment |>
  model(Stl = STL(Employed ~ season(window = 25))) |>
  components()
stl_dcm6 |> autoplot()

stl_dcm7 <- us_retail_employment |>
  model(Stl = STL(Employed ~ season(window = "periodic"))) |>
  components()
stl_dcm7 |> autoplot()
```

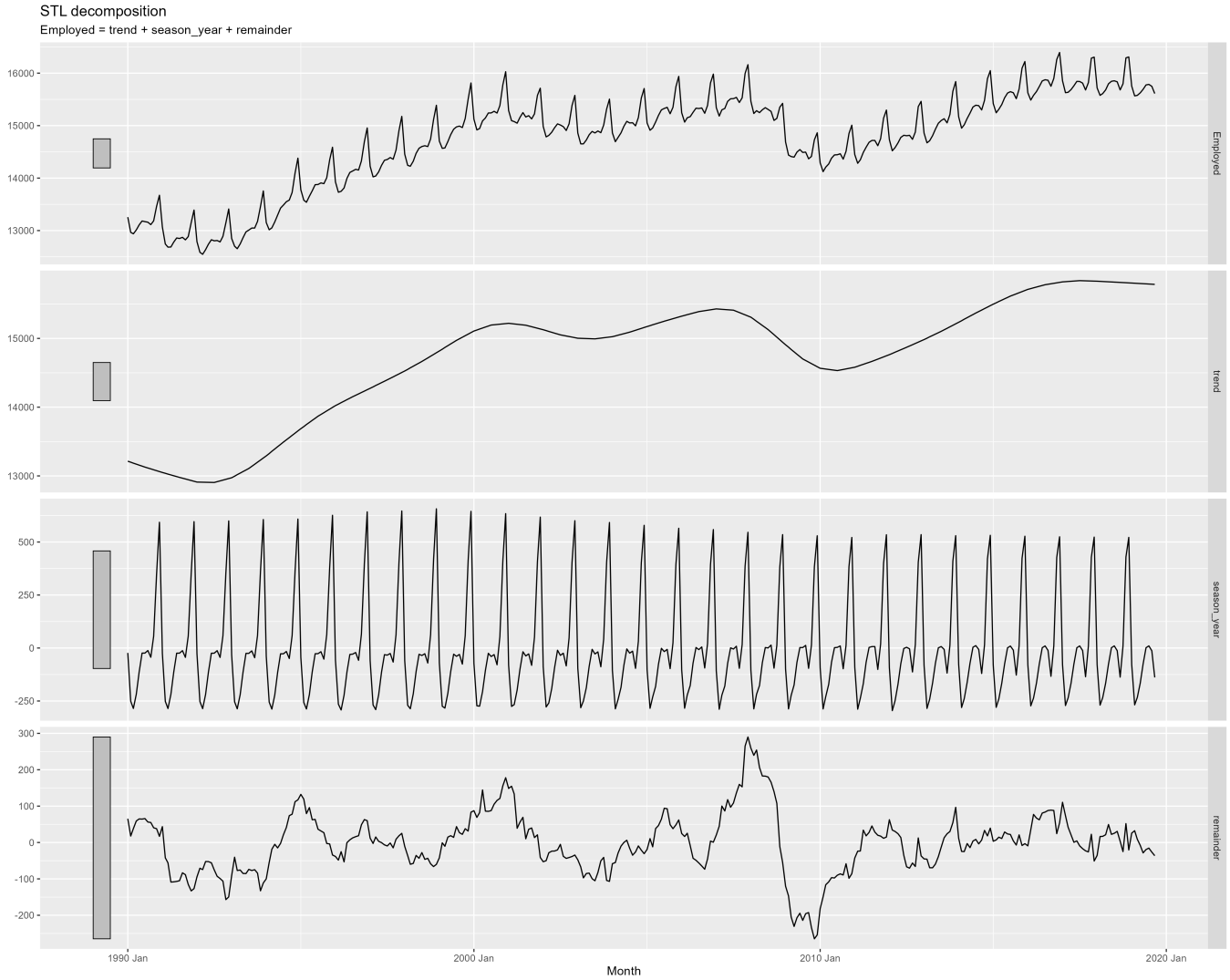
- STL decomposition with default parameters



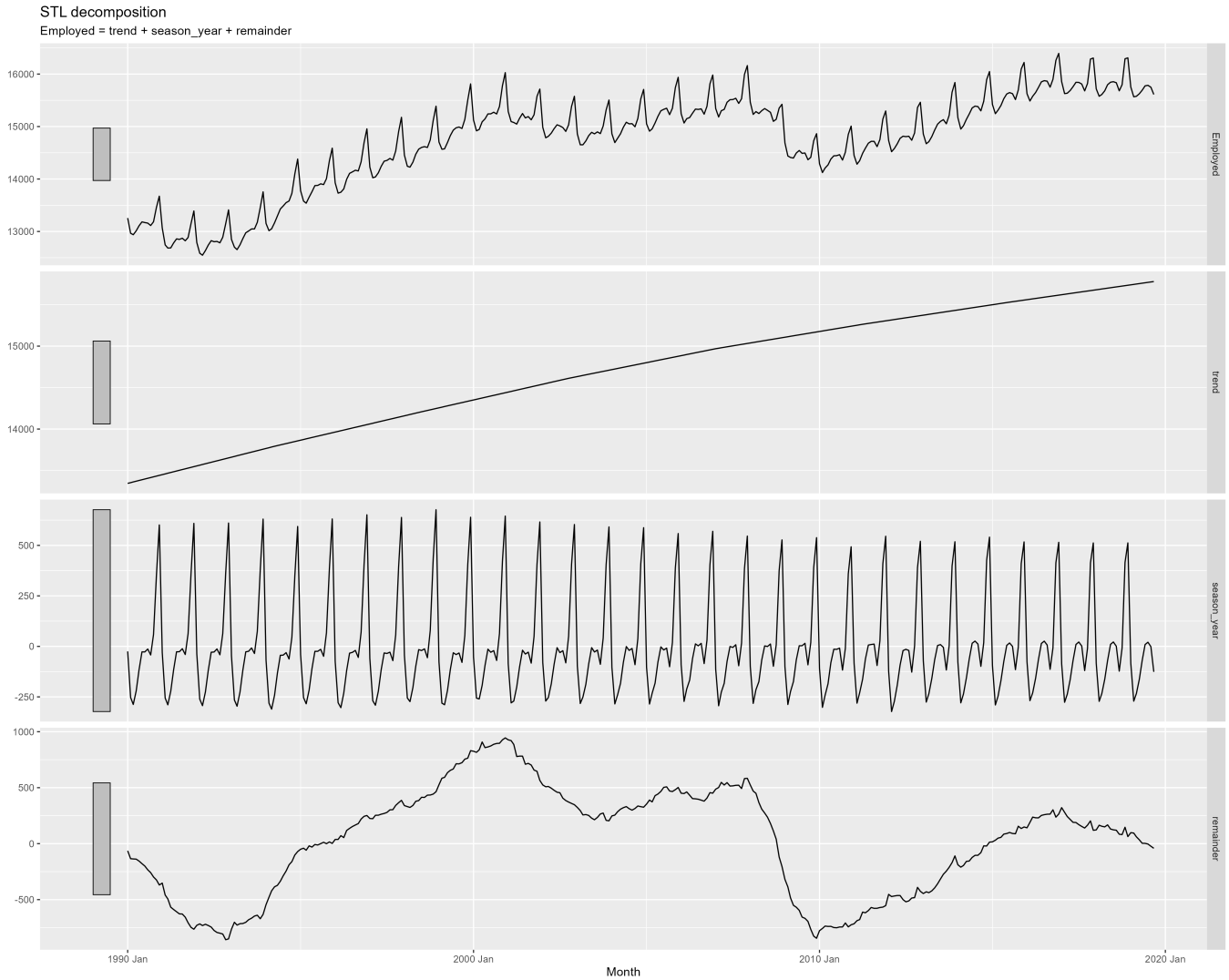
- STL decomposition with trend window = 5



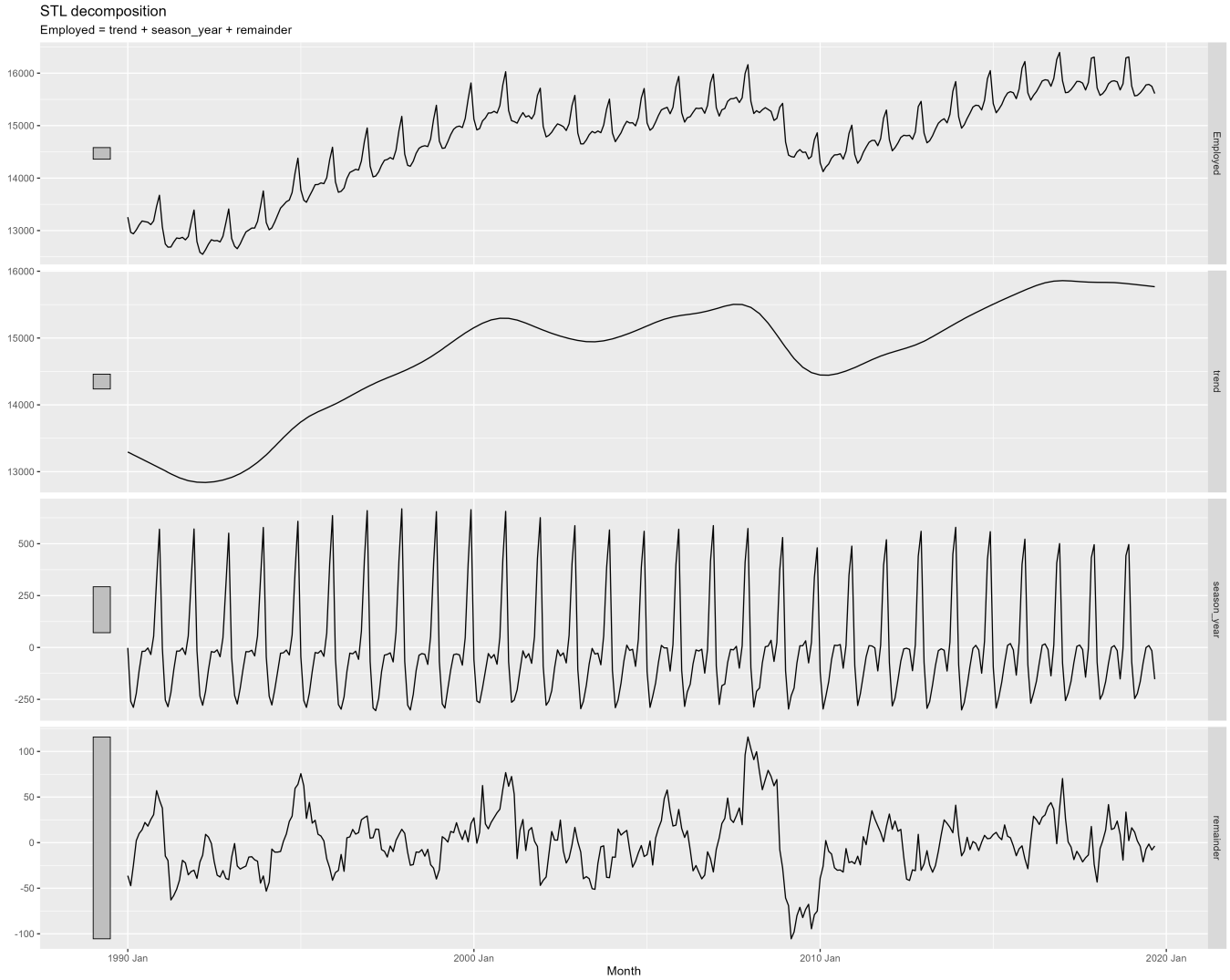
- STL decomposition with trend window = 51



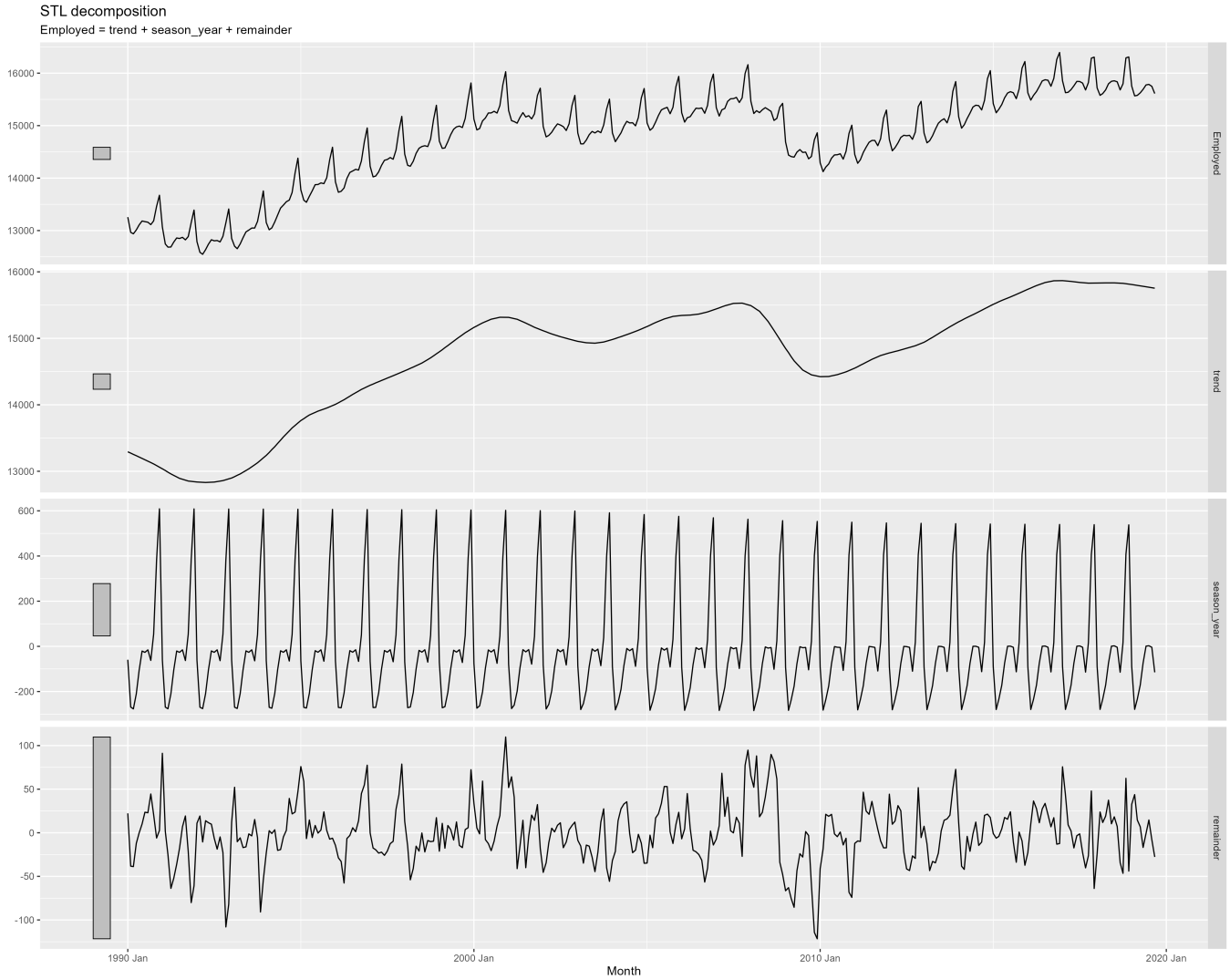
- STL decomposition with trend window = 501 (more than the number of observations)



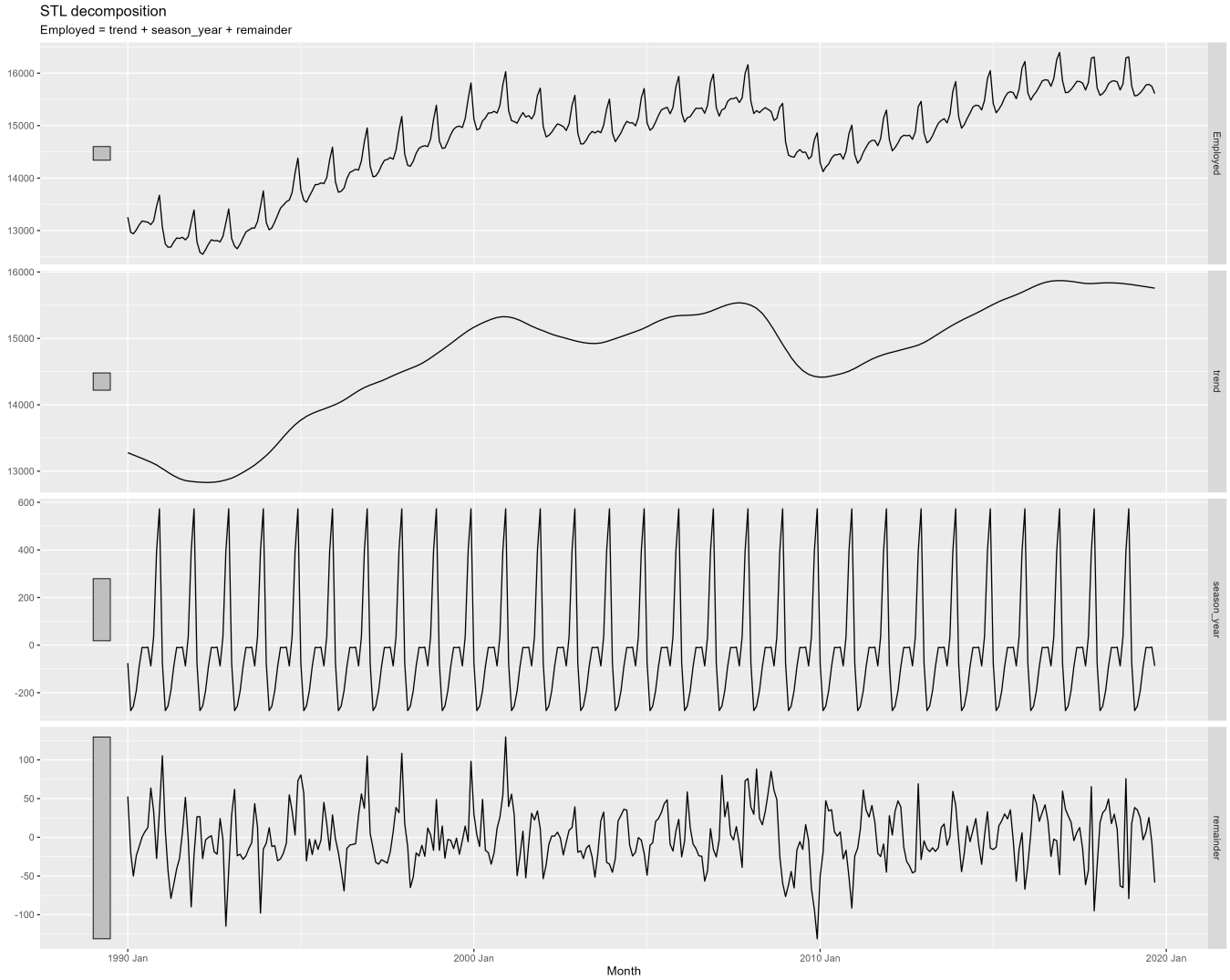
- STL decomposition with season window = 5



- STL decomposition with season window = 25



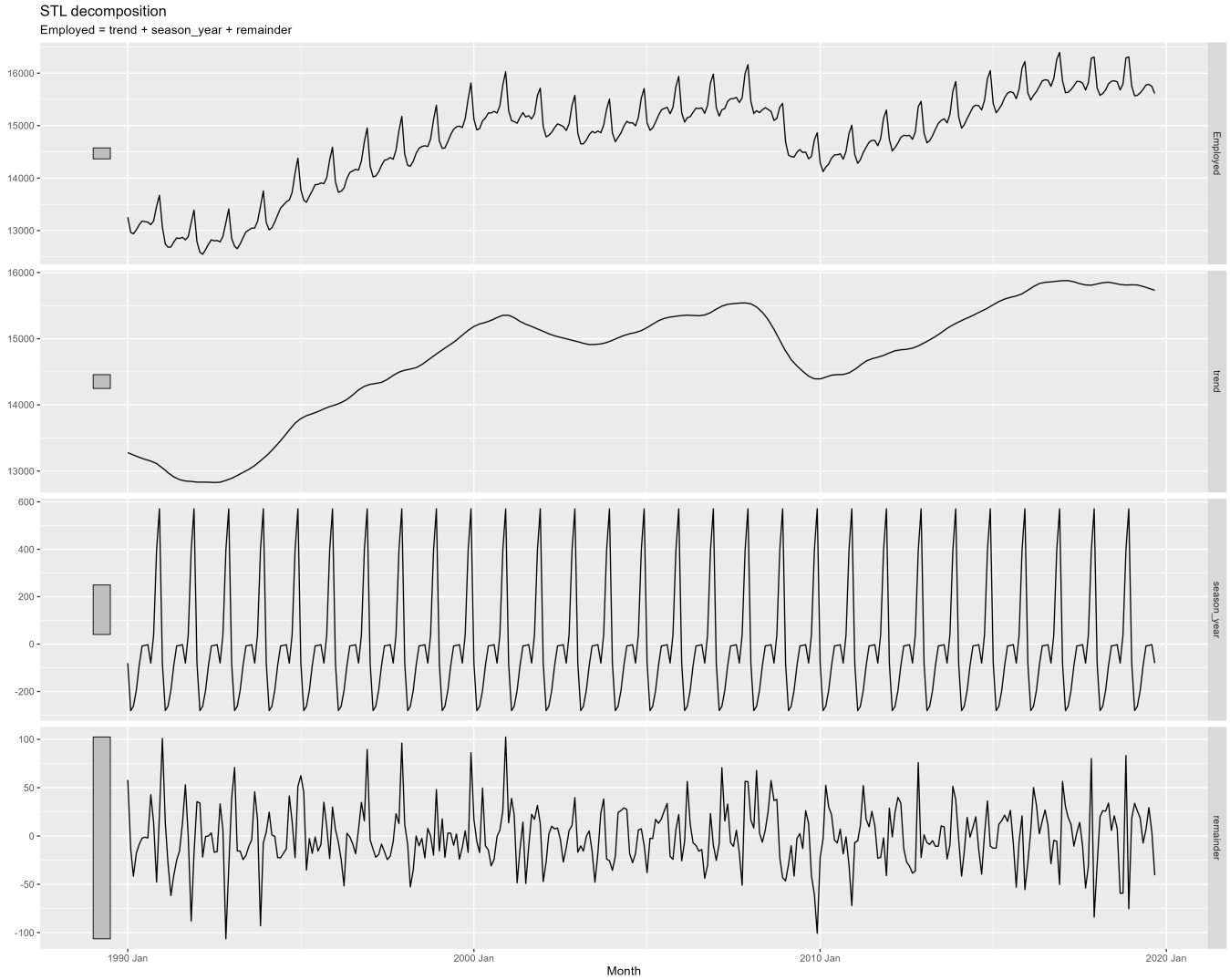
- STL decomposition with season window = "periodic"



- Obviously, we can specify both the trend and the season window in the same time.

```
stl_dcm8 <- us_retail_employment |>
  model(Stl = STL(Employed ~ trend(window = 13) + season(window = "periodic")))
    |>
  components()
stl_dcm8 |> autoplot()
```

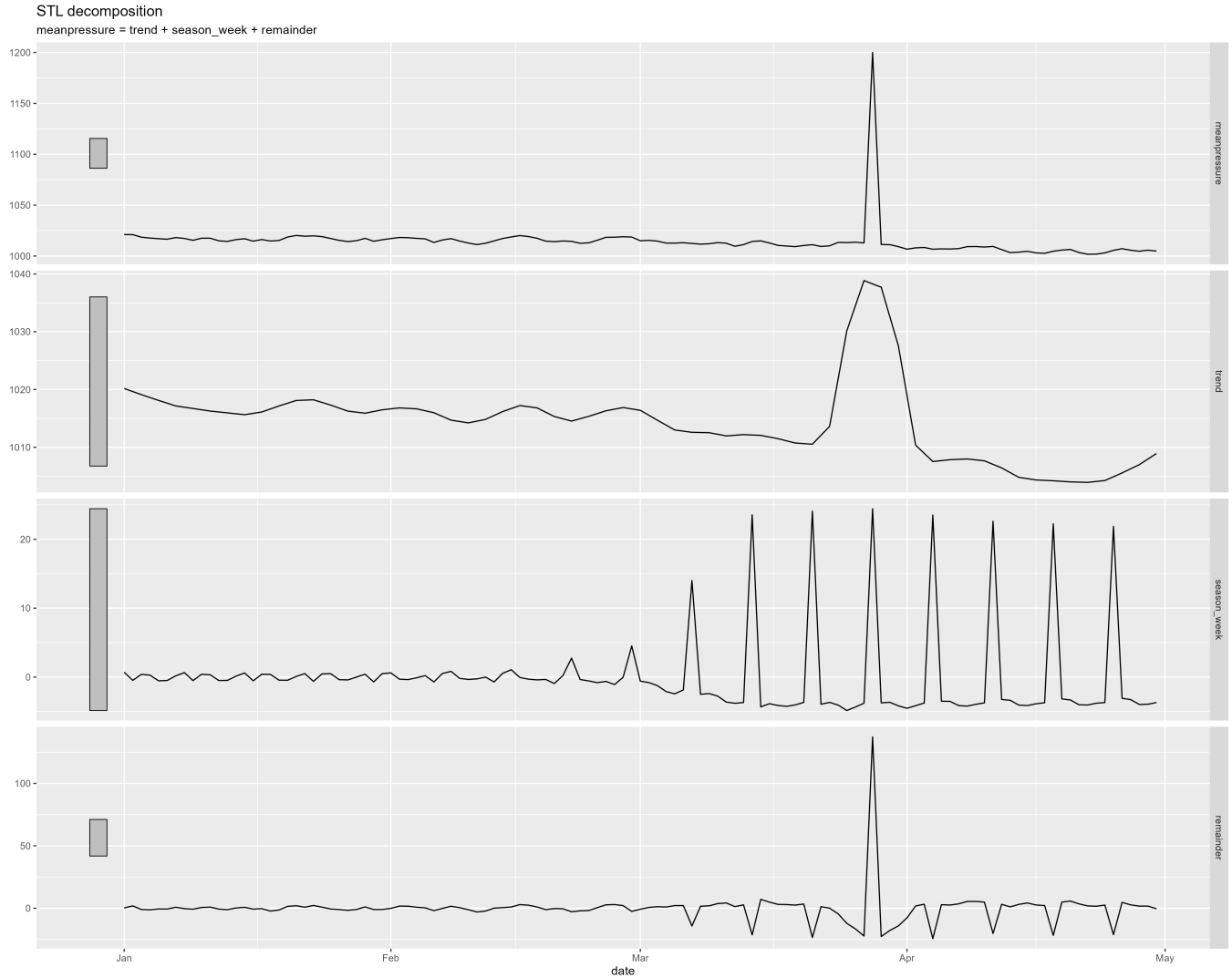
- Season window = periodic and trend window = 13



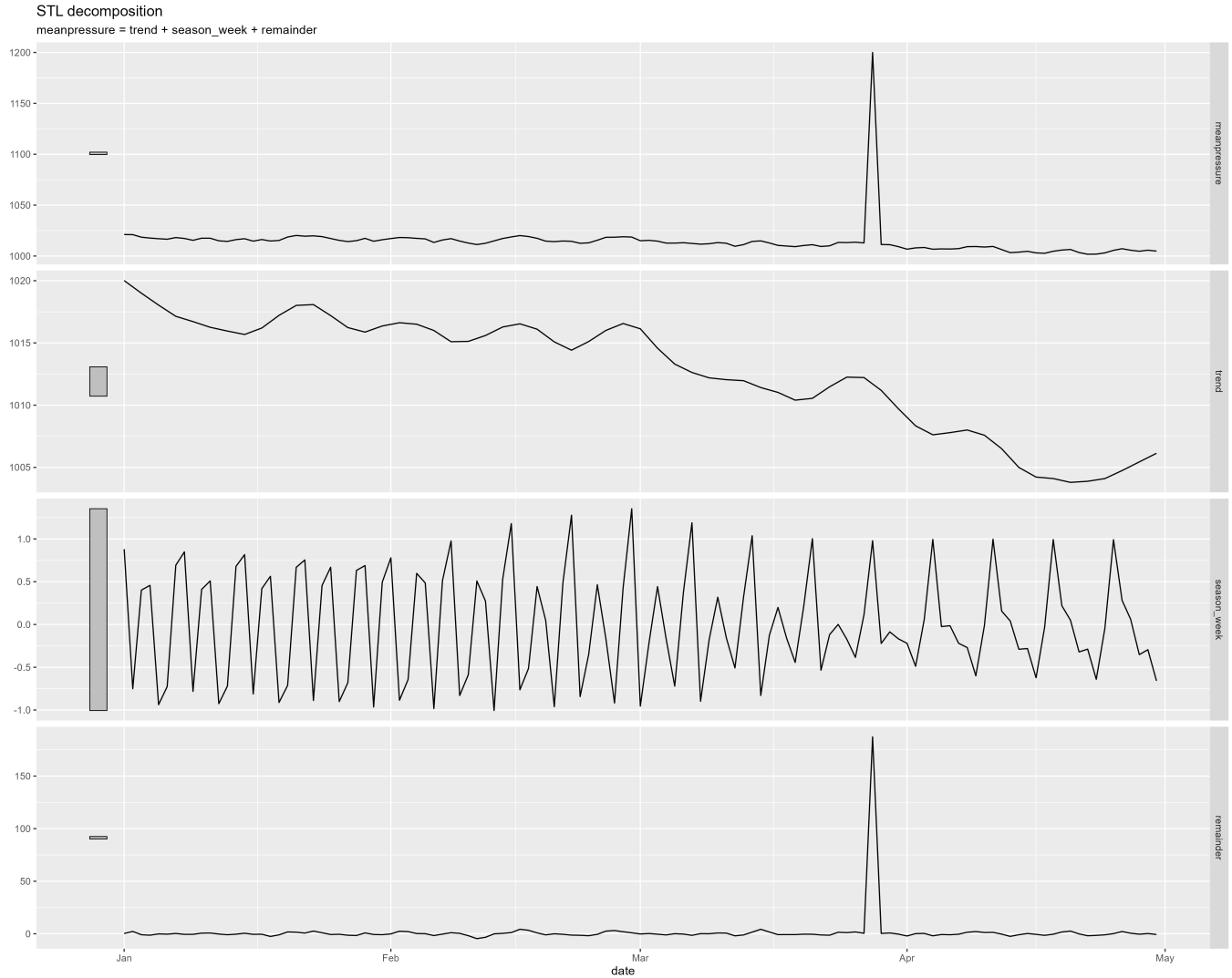
- There are two more parameters that can be set for the STL decompositon in R, which do not influence the trend or season directly: the number of iterations and whether to use a robust decomposition.

```
stl_climate <- climate_ts_short |>
  model(Stl = STL(meanpressure)) |> components()
stl_climate |> autoplot()

stl_climate2 <- climate_ts_short |>
  model(Stl = STL(meanpressure, robust = TRUE)) |> components()
stl_climate2 |> autoplot()
```

- Default STL decomposition



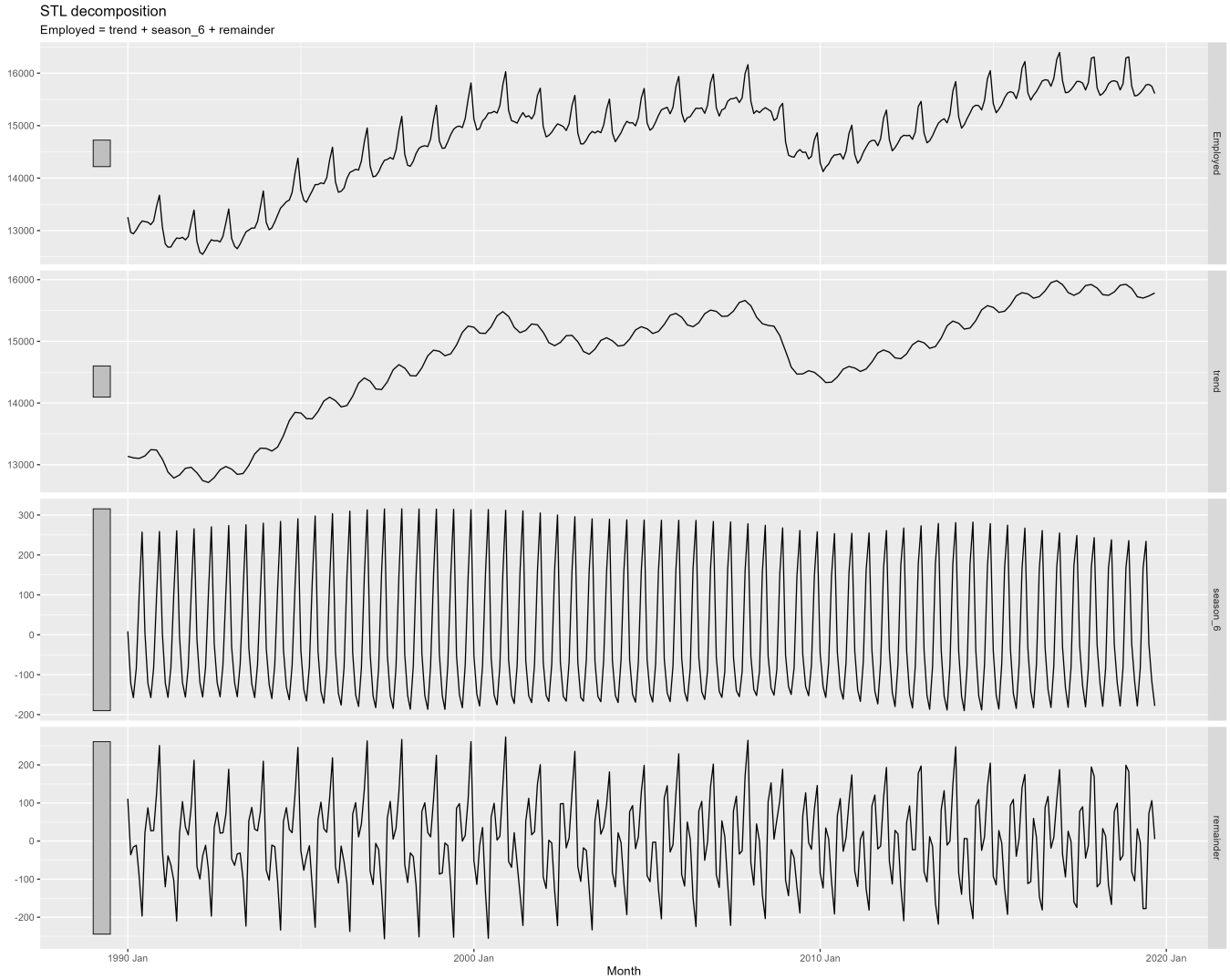
- Robust STL decomposition



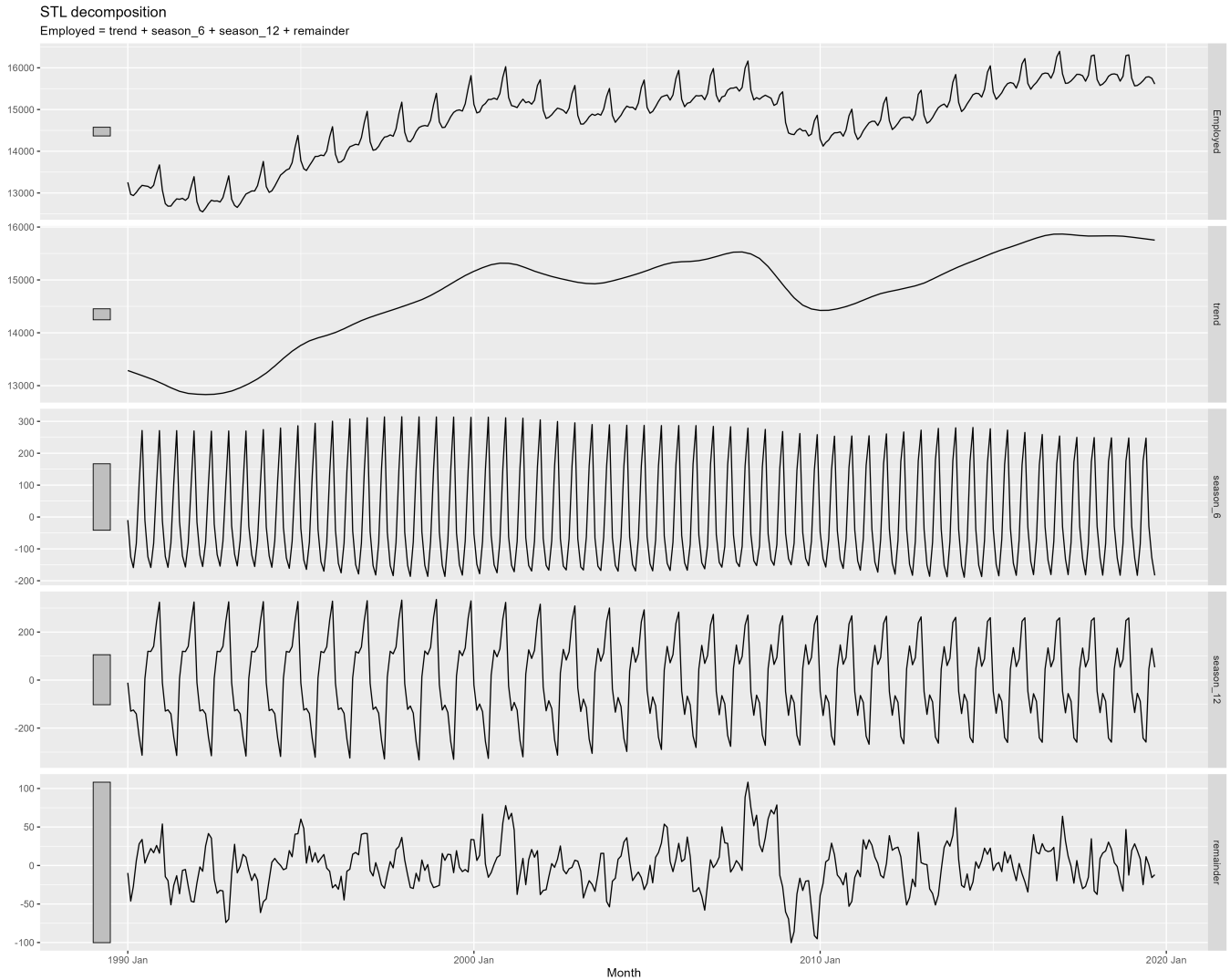
- For the seasonal special, besides the window size we can specify the period as well.
- We can also specify a list of periods.
- Let us see a few examples (for this time series, we do not have multiple seasonalities, just the yearly one, so the following examples have the goal of understanding the decomposition, this is not something you do in real life).

```
stl_dcm11 <- us_retail_employment |>
  model(stl = STL(Employed ~ season(period = 6))) |> components()
stl_dcm11 |> autoplot()
stl_dcm11 |> ACF(remainder) |> autoplot()

stl_dcm12 <- us_retail_employment |>
  model(stl = STL(Employed ~ season(period = c(6, 12)))) |> components()
stl_dcm12 |> autoplot()
```

- STL decomposition with a seasonal period of 6



- STL decomposition with a seasonal period of 6 and one period of 12



Multiple Seasonal-Trend Decomposition using Loess (MSTL)

- It is an extension of the STL algorithm which can decompose data with multiple seasonalities.
- The method was proposed in: K. Bandara, R. J. Hyndman, C. Bergmeir: *MSTL: A Seasonal-Trend Decomposition Algorithm for Time Series with Multiple Seasonal Patterns*, 2021. `https://arxiv.org/abs/2107.13462`
- Since it can have several seasonal components, MSTL assumes that a time series can be decomposed as:

$$y_t = \hat{T}_t + \hat{S}_t^1 + \hat{S}_t^2 + \cdots + \hat{S}_t^n + R_t$$

- where the seasonal components represent different seasonalities (daily, weekly, monthly, etc.)

- The paper which introduced MSTL compared it to other existing methods capable of extracting multiple seasonalities and found that:
  - It is accurate (more accurate than the others on a set of benchmark time series)
  - Computationally efficient
  - Robust (by using the robust flag passed to STL).

Short description of the MSTL algorithm I

- MSTL receives as parameter the lengths of the seasonal components (the periods). For every period (starting from the shortest one and going to the longest one) we extract the seasonal components using STL and then subtract it from the time series.
- The seasonally adjusted time series from the previous step is then passed on, to continue extracting the next seasonal component (previous step is repeated with another period).
- At the end of the previous step we have estimates for a series of seasonalities and the fully seasonally adjusted time series (a time series from which all seasonal components were removed).

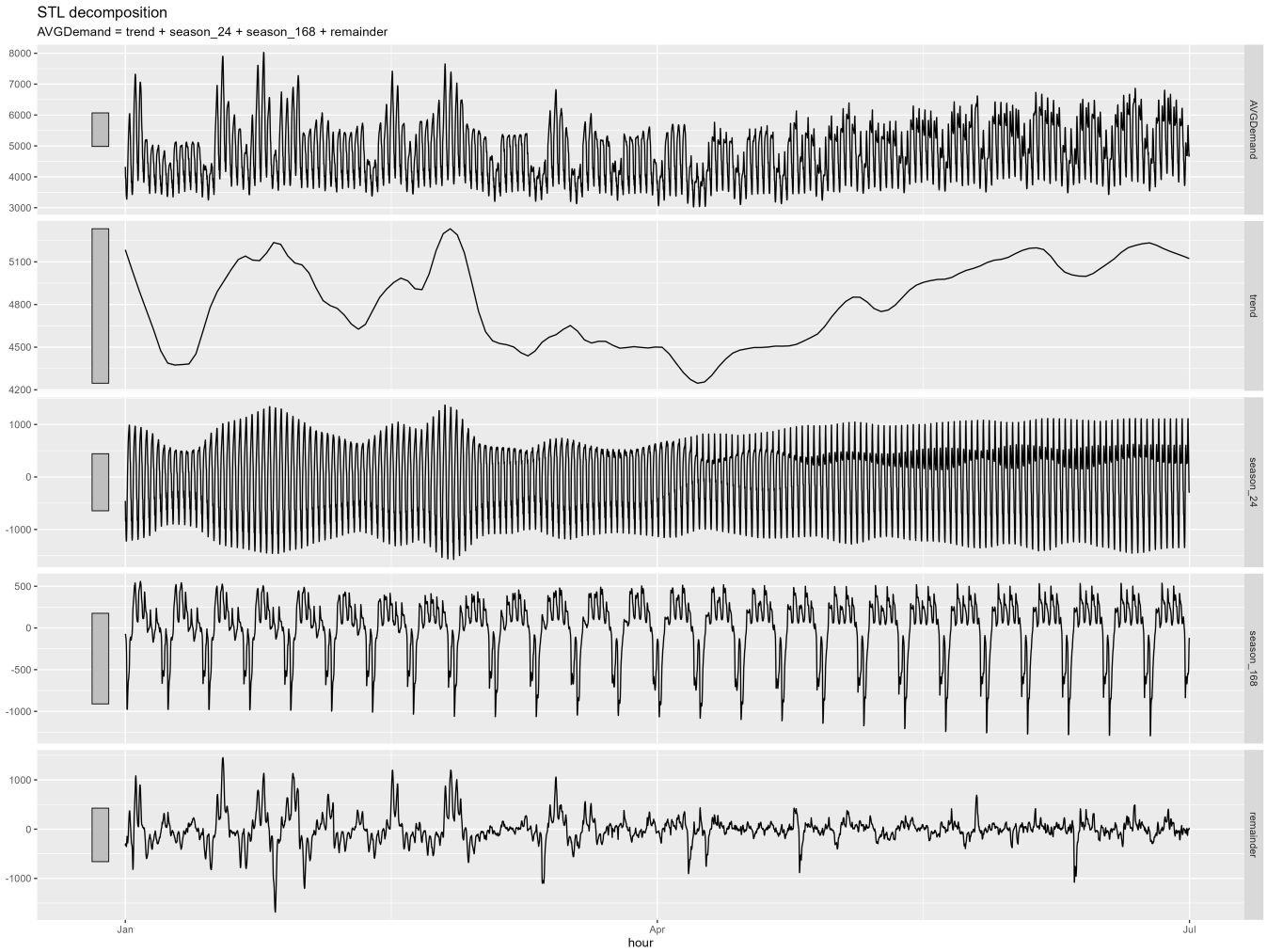
Short description of the MSTL algorithm II

- In the next step we iterate through all seasonal components again, one-by-one, and at every iteration we add back that single seasonal component to the fully seasonally adjusted time series and re-extract it, using STL. Subtract it from the time series, to have again the fully de-seasonalised version.
- Repeat the previous step N times.
- Extract the trend from the last STL fit from the previous version (the one with the longest seasonal period component).
- Extract the residual by subtracting the trend from the de-seasonalised time series.

- The STL function from R that we have used previously, actually implements this MSTL algorithm.

```
vic_elec
vic_elec |>
  index_by(hour = floor_date(Time, "hour")) |>
  summarize(AVGDemand = mean(Demand)) -> vic_elec_daily

vic_elec_daily
vic_elec_daily <- vic_elec_daily |>
  filter(year(hour) == 2012, month(hour) <= 6)

vic_elec_daily |> autoplot()

vic_elec_daily |>
    model(mstl = STL(AVGDemand ~ season(period = c(24, 24*7)))) |>
    components() |>
    autoplot()
```

- MSTL decomposition for VicElec data.



Time series features I

- A feature is simply one value, which describes some property of the time series.
- We can compute different features for time series, which can help us explore the properties of the time series.
    - We have already seen some features (and we have actually used the *features* function) before: the Guerrero estimate for the Box-Cox transformations, the number of times regular or seasonal differencing should be applied to make a time series stationary, the autocorrelations, etc.
- For exploring different features, we will use as example the Australian quarterly tourism data, found in the *tourism* tsibble.

Time series features II

- We have simple features: minimum, maximum, mean. The *features* function will compute them. In case of tsibbles containing several time series, it will compute the value for each time series.
- Function *quantile* provides the five summary statistics: minimum, first quartile, median, third quartile and maximum.

```
tourism |>
  features(Trips, list(Mean = mean)) |>
  arrange(Mean) #to sort increasingly by mean

tourism |>
 features(Trips, quantile)
```

Autocorrelation features I

- Remember, autocorrelation is the correlation between a time series and its lagged version (version with some time steps behind).
- We have talked about the *acf* plot, where autocorrelation values for different lags are visualized.
- But we can compute new numerical features from autocorrelation, for example, the sum of squares for the first 10 autocorrelations is an indicator of how much autocorrelation is in the data in general.
- We can also difference the data and compute autocorrelation in this new series (both for simple and seasonal differences).
- We can difference the already differenced data as well and compute autocorrelation features for this as well.

Autocorrelation features II

- In R we can compute autocorrelation features with the *feat_acf* function which returns the following:
  - the first autocorrelation coefficient of the original data
  - the sum of the squares for the first 10 autocorrelation coefficients of the original data (shows overall how much autocorrelation is in the data).
  - the first autocorrelation coefficient of the differenced data
  - the sum of the squares for the first 10 autocorrelation coefficients of the differenced data
  - the first autocorrelation coefficient of the twice differenced data
  - the sum of the squares for the first 10 autocorrelation coefficients of the twice differenced data
  - for seasonal data the autocorrelation coefficient at the first seasonal lag is also returned

Autocorrelation features III

```
tourism |>
    features(Trips, feat_acf)
```