VE472 Lecture 9

Jing Liu

UM-SJTU Joint Institute

Summer

very large n

- Consider the following large dataset, which has only 3 columns but...

```
> DATA = data.table::fread("large_472.gz")
> str(DATA)
```

```
                X1                X2           Y
         1:   88 -1.746573094   8.087930
         2:   88  0.658447891   8.957405
        ---
299999999:   96 -0.218207636   8.875289
300000000:  108  0.004935685  10.489749
```

- Notice the original data file is in a compressed format, gzip in this case,

```
> file.info("large_472.gz")$size / 10^9
```

```
[1] 5.347417
```

when it is uncompressed and read into the memory, the data is even bigger.

```
> pryr::object_size(DATA)
```

```
6 GB
```

- Since most computers nowadays still have fewer than 16GB of memory,

```
> large.lm = lm(Y~., data = DATA)
```

```
Error: vector memory exhausted (limit reached?)
```

any dataset bigger than 4GB, is too big to be analysed in the usual way.

- Note the computational as well memory requirement prohibit us in doing so.
- Especially, if we want to consider a more flexible model,

$$Y = f_1(X_1) + f_2(X_2) + \varepsilon$$

where $\varepsilon \sim \mathrm{N}(0, 1)$, f_1 and f_2 are smooth functions, then the computational cost is going to be an even bigger issue than the memory requirement.

- Suppose all data points are independent, given the dimension of the problem is very low, $k = 2$, we do not really need all those 300 million data points.

- In this case, a simple random sampling scheme will actually be sufficient.

```
> (total.row.n = nrow(DATA))
```

```
[1] 300000000
```

- Consider using 5000 data points instead of 300 million

```
> sample.n = 5000L
> index = sample(1:total.row.n, sample.n)
> index[1:3]
```

读进来再sample

```
[1] 272402730 186655906 115527201
```

```
> DATA.sample = DATA[index, ]
> nrow(DATA.sample)
```

```
[1] 5000
```

- Of course, it would be better to perform a simple random sampling scheme without loading the whole dataset into memory, which may not be possible.

```
> rm(list=ls())
> sample.n = 5000L

> # Find how many rows the data has without header
> row.n = R.utils::countLines("large_472.gz")-1

> # Sample the required number of rows
> index = sample(1:row.n, sample.n)
> index = index[order(index)]
> skip.vec = c(index[1], diff(index)-1)
                          每次要skip多少行
> # Create an object to put the data
> DATA.sample = data.table::data.table(
+                         X1 = integer(sample.n),
+                         X2 = double(sample.n),
+                         Y=double(sample.n))
```

```
> # Open a connection to the data file
> con = file("large_472.gz", open = "r")
>
> # Load the required data points
> for (i in 1:sample.n){
+   skip = skip.vec[i]
+   data.tmp = scan(
+     con, nlines = 1, skip = skip,
+     sep = ",", quiet = TRUE,
+     what = list(integer(), double(), double()))
+   DATA.sample[i, (1:3):= data.tmp]
+ }
> close(con)

> DATA DATA.sample
```

```
        X1           X2          Y
   1:   87   0.6907444   8.499598
  ---
5000:  101  -0.8642063   8.767105
```

- Loading the whole dataset without doing anything else need

```
> system.time({
+   DATA = data.table::fread("large_472.gz")
+ })
```

```
   user  system elapsed
 73.122  27.769 100.598
```

while loading 5000 lines using the method on the last page need

```
   user  system elapsed
431.998   2.323 145.937
```

- So it is not for speeding things up but addressing the limited memory issue.

```
> pryr::object_size(DATA.sample)
```

```
101 kB
```

- The rest becomes a standard linear smoother (local regression) problem

local polynomial regression

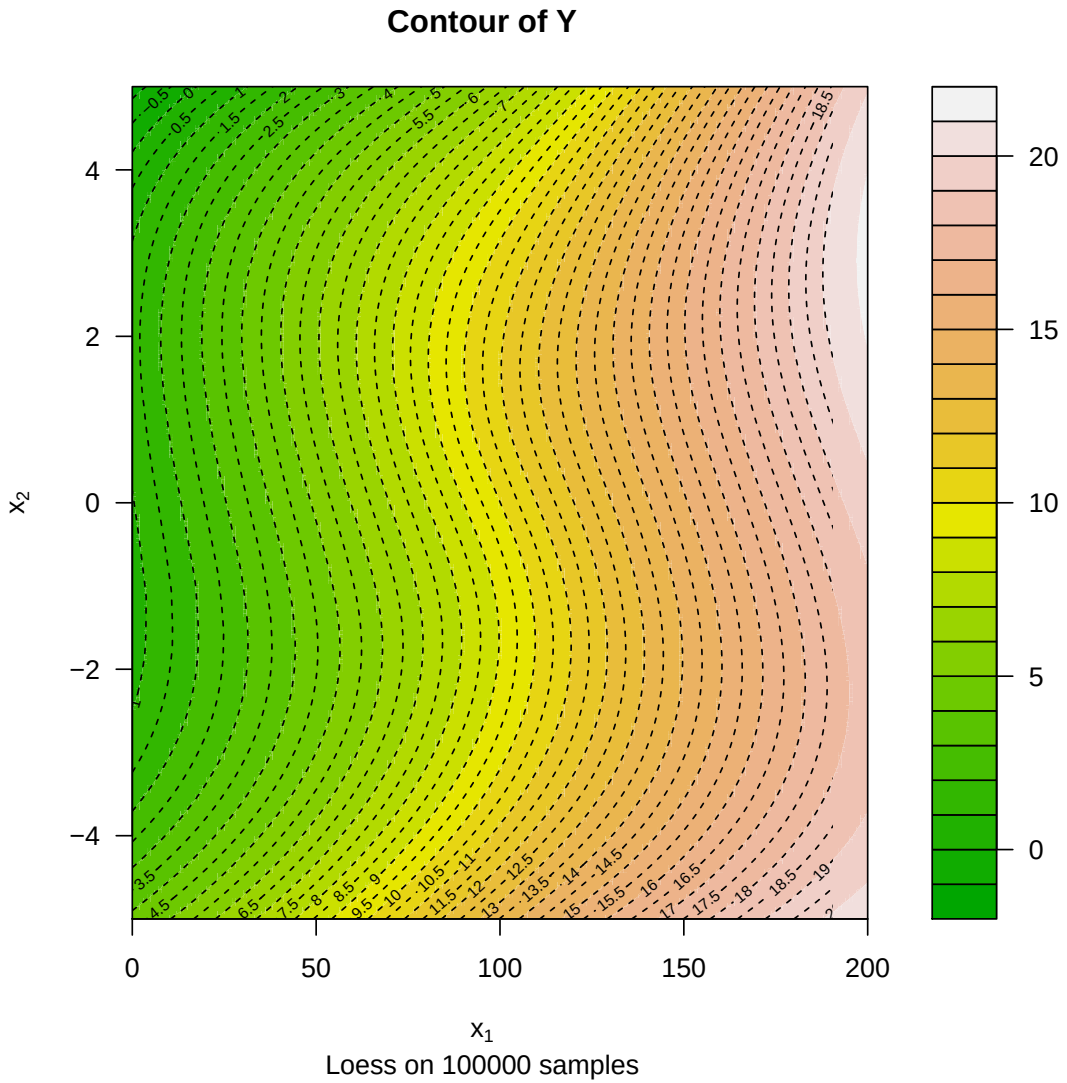
```
> model.lo = loess(
+   Y~X1+X2, control=loess.control(surface="direct"),
+   data = DATA.sample)

> # Create the range of X1 and X2 values
> x1.n = 200
> x2.n = 200
> x1.plot = seq(0, 200, length.out = x1.n)
> x2.plot = seq(-5, 5, length.out = x2.n)
> grid = expand.grid(x1.plot, x2.plot)
>
> newdata = data.frame(grid)
> names(newdata) = c("X1", "X2")
>

> # Fitted values for this nonlinear surface
> Y = predict(model.lo, newdata = newdata)
> Y = matrix(Y, nrow = x1.n, ncol = x2.n)
```

The model gradually becomes stable when $n \to 1e5$

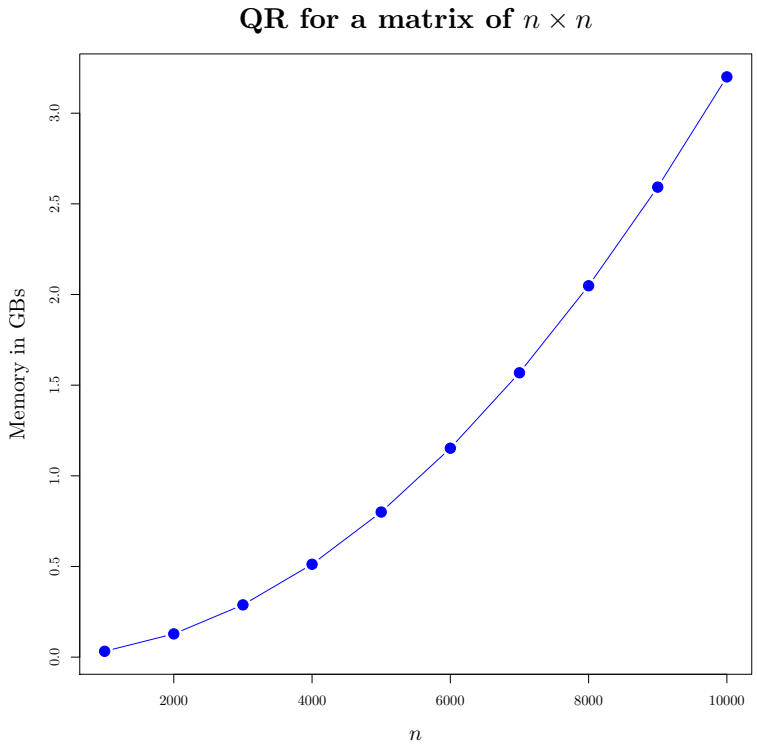
converge



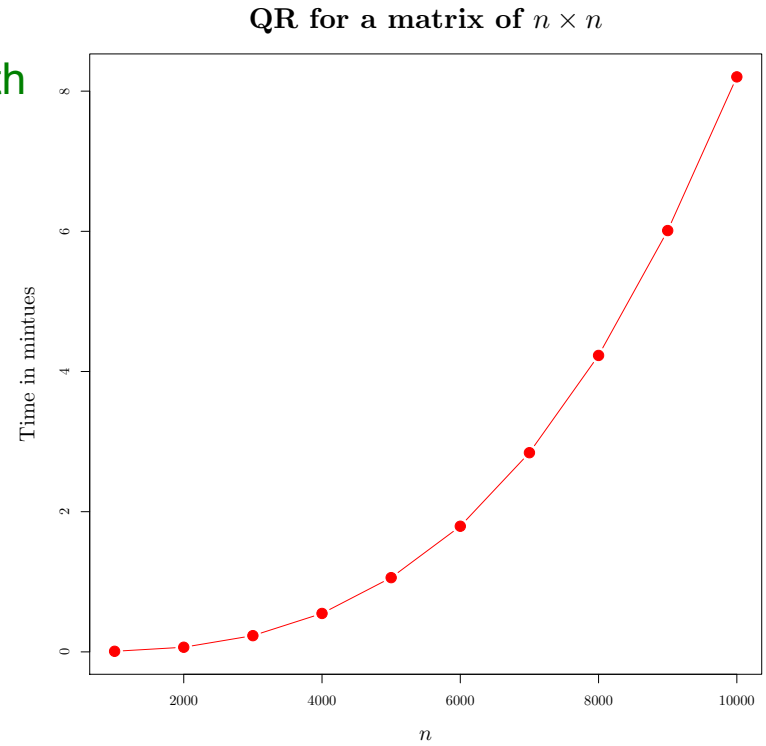
- Basing on 1 million sample points out of those 300 million data points will take hours! So the CPU bound actually kicks in before the memory bound.
- Since common methods in data science often involve the followings,

  Choleksy, QR, SVD or numerical optimisation/integration

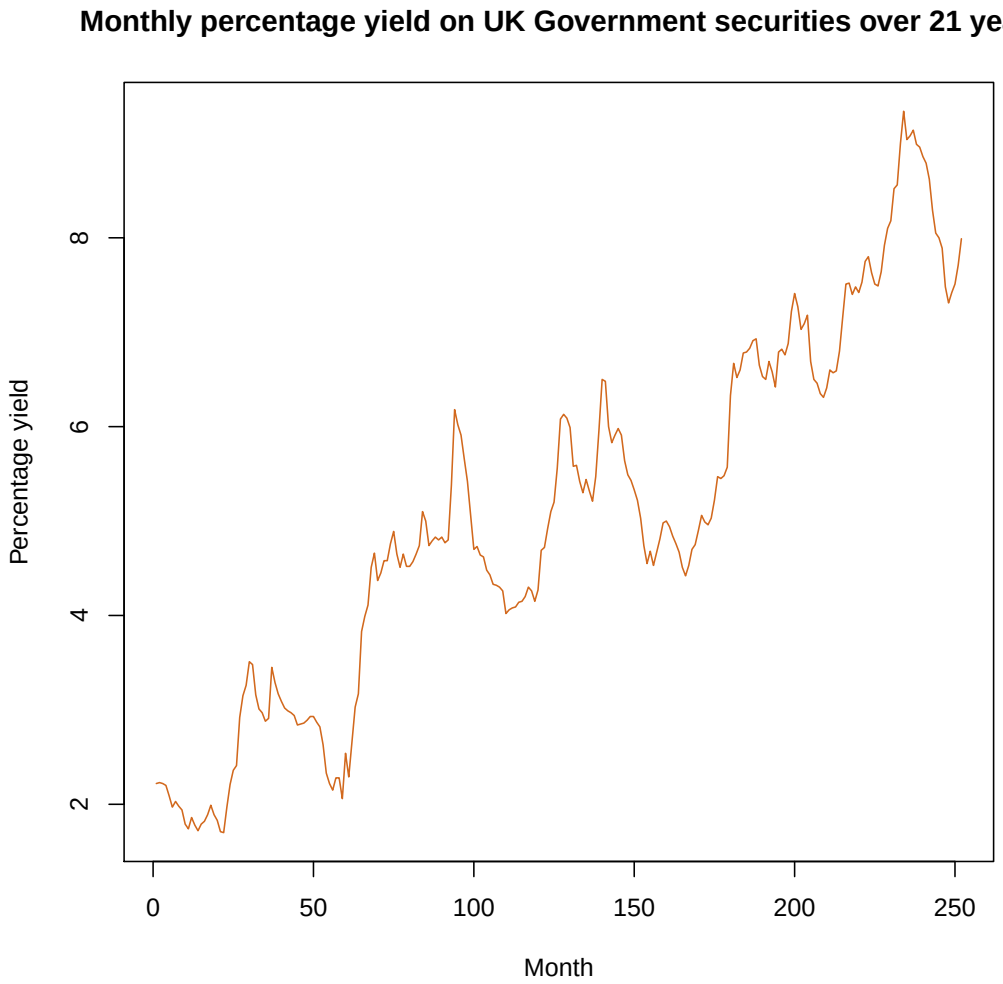
  knowing the limits of those algorithms will give you some roughly idea on whether some kind of sampling scheme is needed for your dataset. e.g.



fast growth



- Although a simple random sampling scheme is effective in many problems, it is not always appropriate even for low dimensional problems. For example



- Time series arise as recordings of processes which vary over time.

$$Y_1, \ldots, Y_T$$

correlated

- The data points in a time series are usually not independent, thus containing information to predict nearby values, potentially each value connected to the next, thus a simple random sampling scheme destroys this structure!
- Another big data issue regarding time series is there are a huge collection

serveral time serires

$$\begin{matrix} Y_{11}, Y_{12} \cdots\cdots Y_{1T_1} \\ Y_{21}, Y_{22} \cdots\cdots Y_{2T_2} \\ Y_{i1}, Y_{i2} \cdots\cdots \quad \vdots \\ \vdots \quad \ddots \ddots \ddots \quad \vdots \\ Y_{n1}, Y_{n2} \cdots\cdots Y_{nT_n} \end{matrix}$$

that is, n, T_1, T_2, ..., T_n could all be very large.

- Traditionally, the followings are the two main interests in time series analysis:

  Smoothing:

  The observed Y_t are assumed to be the result of "noise" ε_t a signal η_t,

  $$Y_t = \eta_t + \varepsilon_t$$

  remove the noise

  we may wish to recover the values of the underlying η_t.

  Forcasting: Forecasting

  On the basis of observation

  $$Y_1, Y_2, \ldots, Y_T,$$

  we may wish to predict what the value of Y_{T+L} will be $(L \geq 1)$.

- But increasingly there is a need to identify, classify or predict some attribute using a large collect of time series over a period in which the attribute lies in.

- Recall we could deal with time series using linear model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

however, instead of assuming that the errors are independent, we assume

$$\boldsymbol{\varepsilon} \sim \text{Normal}\left(\mathbf{0}, \boldsymbol{\Sigma}\right)$$

where the covariance matrix $\boldsymbol{\Sigma}$ as well as $\boldsymbol{\beta}$ need to be estimated.

- Before considering any fancy model, we could consider decomposing Y_t into several components, and model each individually later on.
- If we assume an additive decomposition, then traditionally we have

$$Y_t = T_t + C_t + S_t + R_t \quad \text{where}$$

T_t represents the long-term increase or decrease in the data.

C_t represents repeated but non-periodic fluctuations in the data.

S_t represents periodic fluctuations in the data.

R_t represents noise/remainder in the data.

cycle

no constant period

波动

- Instead having a simple T_t and C_t, we could combine the two,

$$Y_t = T_t + S_t + R_t \quad \text{where}$$

where T_t captures the non-periodic fluctuations and the long-term trend.

- Alternatively, a multiplicative decomposition is given by

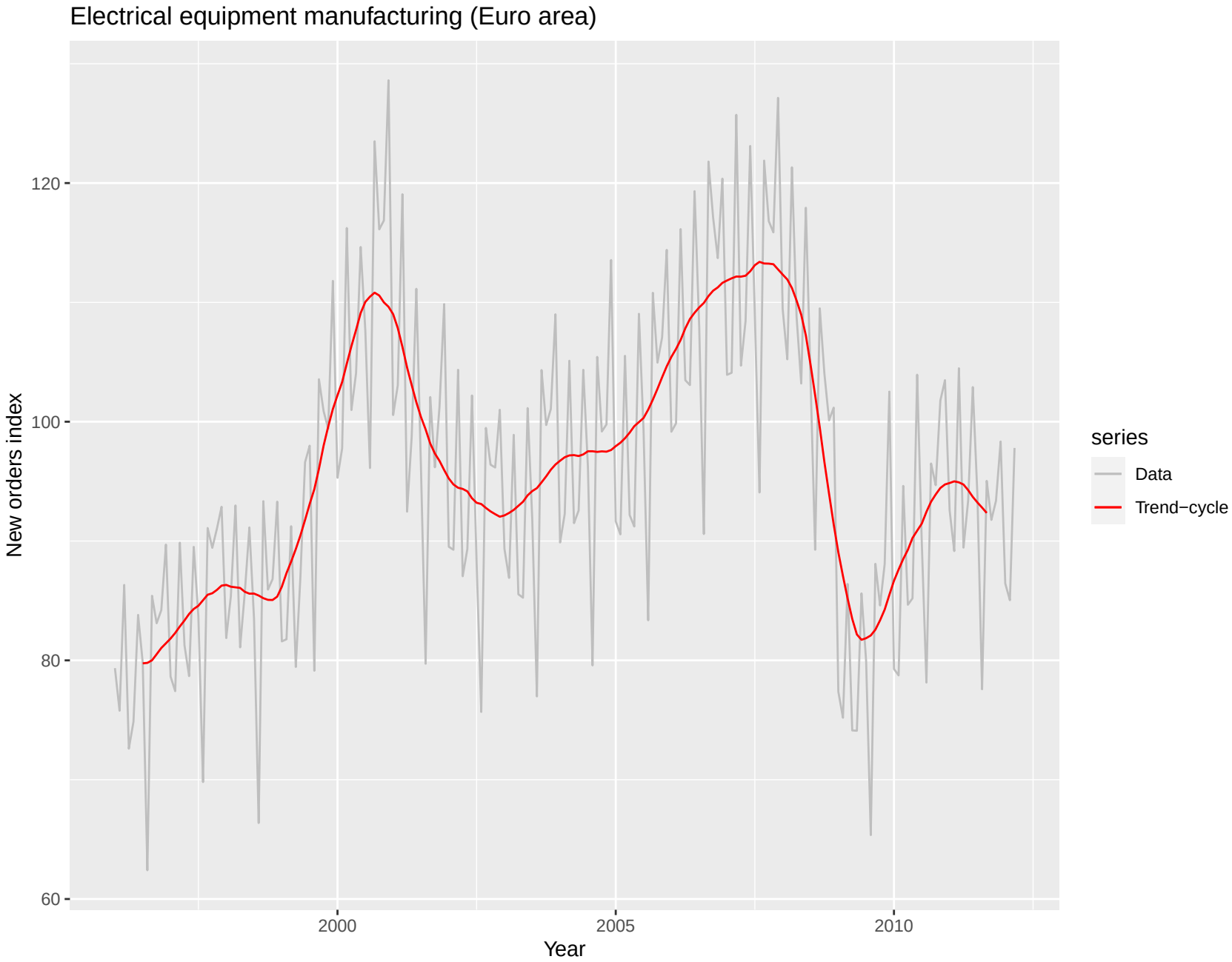
$$Y_t = T_t \times S_t \times R_t$$

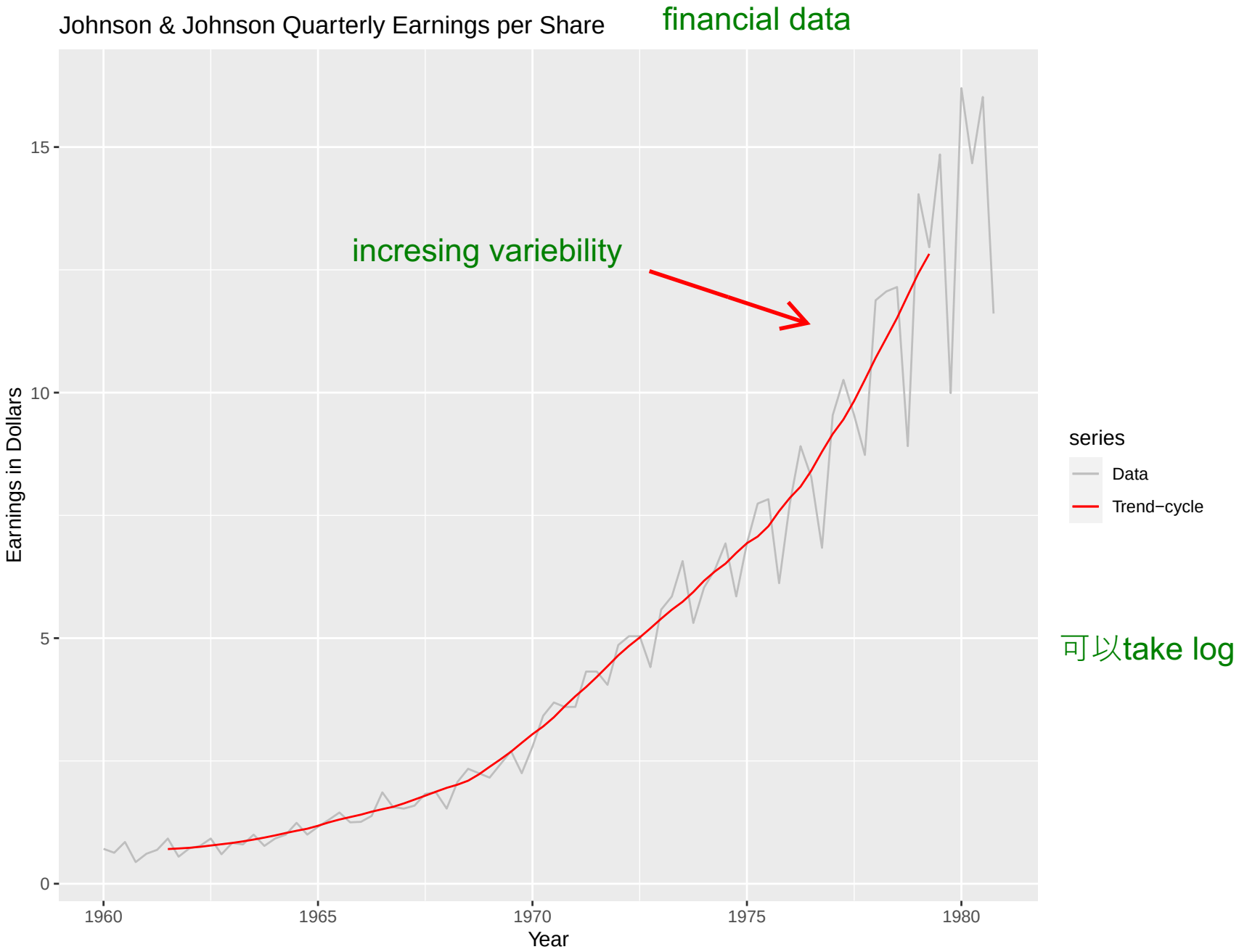
as time increses, more variebility appears

- The additive decomposition is usually if the magnitude of S_t or the variation of T_t does not vary as t increases. If not, the multiplicative decomposition or a log transformation is needed to stabilise the variability over time

$$\log Y_t = \log T_t + \log S_t + \log R_t$$

- Multiplicative decompositions are common with financial time series.


Electrical equipment manufacturing (Euro area)
New orders index
120
100
80
60
2000
2005
2010
Year
series
Data
Trend−cycle


Johnson & Johnson Quarterly Earnings per Share
financial data
incresing variebility
Earnings in Dollars
15
10
5
0
1960
1965
1970
1975
1980
Year
series
Data
Trend−cycle
可以take log

Moving average smoothing

- Recall a moving average of order m can be written as

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j} \qquad \text{where } m = 2k+1.$$

average around t

- It gives an estimate of the T_t by averaging values of the time series nearby.
- The idea is nearby values are under the same periodic force, and averaging eliminates, at least to some extent, the randomness due to S_t and R_t, thus leaving only trend-cycle component.
- Of course, there are more sophisticated smoothing, for example,

2×4-MA :

weighted moving averge. Nearby has more weight

$$\begin{aligned}\hat{T}_t &= \frac{1}{2}\left(\frac{1}{4}\left(y_{t-2}+y_{t-1}+y_t+y_{t+1}\right)+\frac{1}{4}\left(y_{t-1}+y_t+y_{t+1}+y_{t+2}\right)\right) \\ &= \frac{1}{8}y_{t-2}+\frac{1}{4}y_{t-1}+\frac{1}{4}y_t+\frac{1}{4}y_{t+1}+\frac{1}{8}y_{t+2}\end{aligned}$$

Classical Decomposition

Step 1

Compute $\hat{T}_t$ using linear or nonlinear smoother.

Step 2

Compute the detrended series: $y_t - \hat{T}_t$ or $y_t / \hat{T}_t$

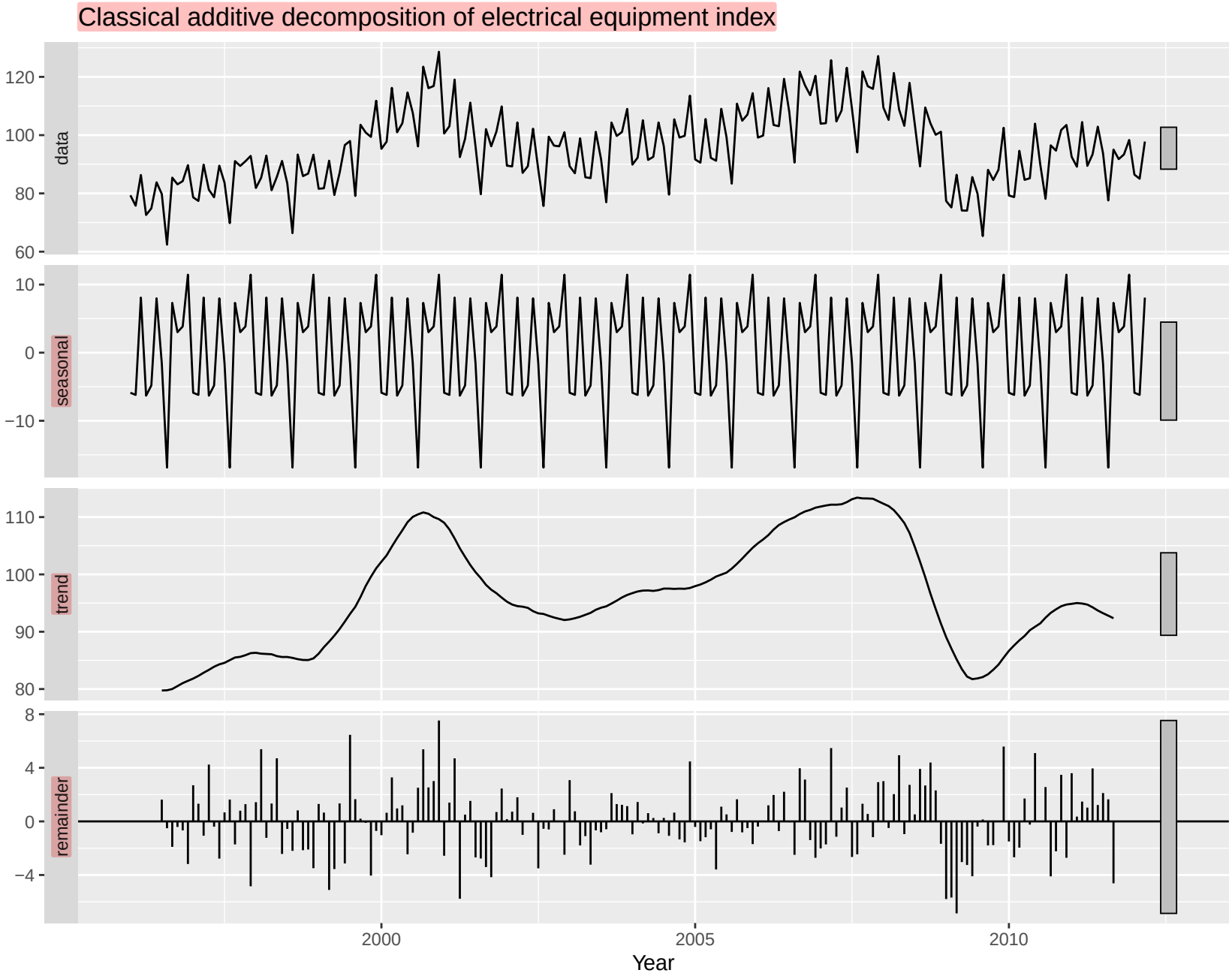
Step 3

Compute $\hat{S}_t$ by simply averaging the detrended values for that season.

Step 4

Compute $\hat{R}_t$ by subtracting/dividing $\hat{T}_t$ and $\hat{S}_t$.

$$R_t = y_t - \hat{T}_t - \hat{S}_t \qquad \text{or} \qquad R_t = y_t / \left(\hat{T}_t \hat{S}_t\right)$$


Classical additive decomposition of electrical equipment index
data
seasonal
trend
remainder
120
100
80
60
10
0
−10
110
100
90
80
8
4
0
−4
2000
2005
2010
Year

SLT decomposition

better than moving average -> local: straight line

- STL stands for "Seasonal and Trend decomposition using Loess", where Loess is local polynomial regression, local in the sense only data points nearby are used.
- It constructs an additive decomposition,

pnly additive

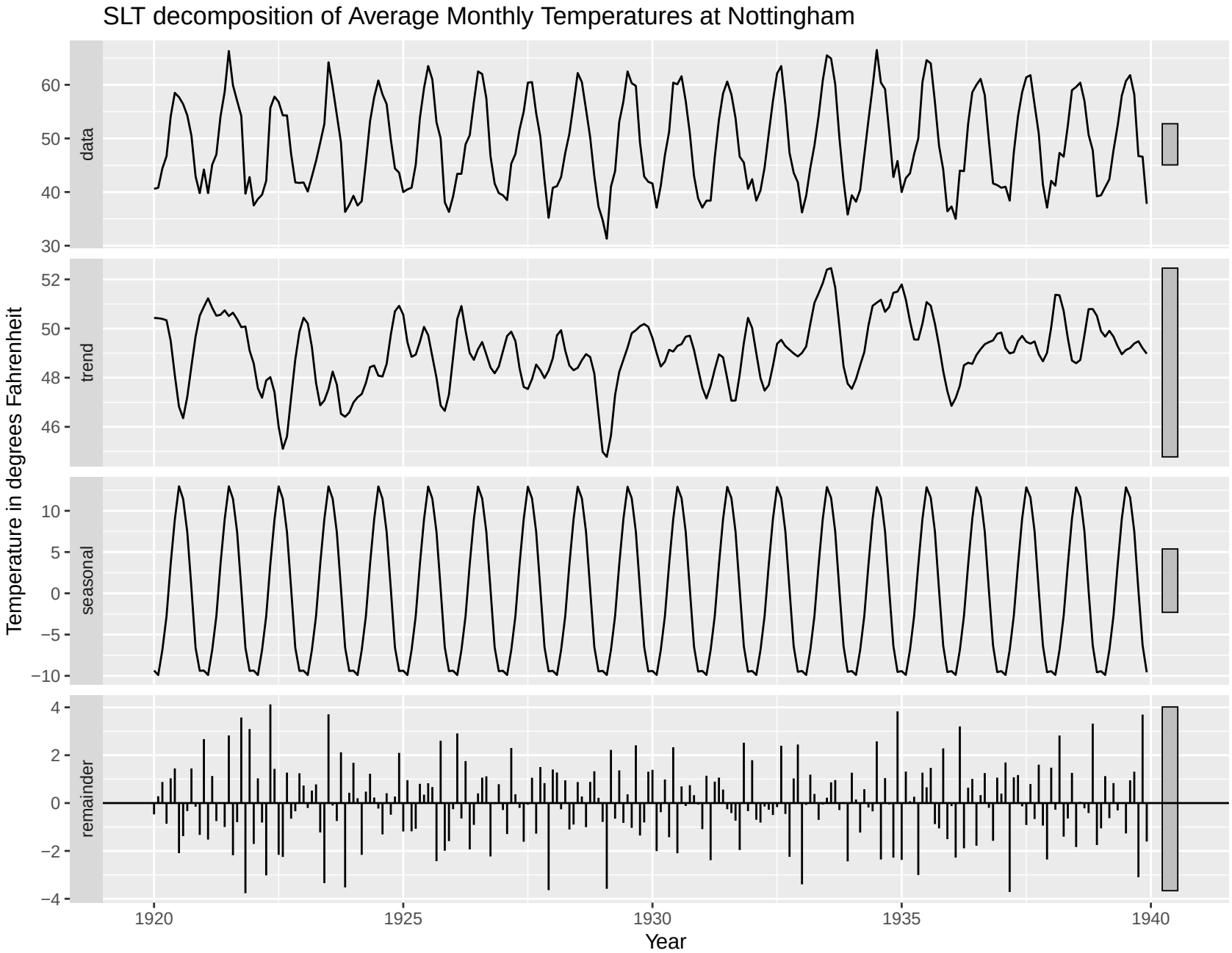
```
> stl(nottem, s.window = 7, t.window = 50)
```

```
 Call:
 stl(x = nottem, s.window = 7, t.window = 50)
```

seasonal trend

```
Components
              seasonal     trend     remainder
Jan 1920   -8.07925728 49.82196 -1.142707381
Feb 1920   -9.25894766 49.78641  0.272538172
```

log-transformation on the data is needed for multiplicative decomposition.


SLT decomposition of Average Monthly Temperatures at Nottingham
data
trend
seasonal
remainder
Temperature in degrees Fahrenheit
Year
60
50
40
30
52
50
48
46
10
5
0
−5
−10
4
2
0
−2
−4
1920
1925
1930
1935
1940

Forecasting with decomposition

- In terms of prediction,

$$Y_t - S_t$$

  is known as the seasonally adjusted component, and

$$S_t$$

  is usually assumed to be fixed for a given season, or changing very slowly.

- The simplest way to predict S_{T+1} is to use $\hat{S}_t$ of the corresponding season if it is fixed, or the last $\hat{S}_t$ of the corresponding season if it is not (SLT).
- So once we $\hat{T}_{T+1}$ from the smoother, we add the seasonal components back

$$\hat{Y}_{T+1} = \hat{T}_{T+1} + \hat{S}_{T+1}$$

STL Example I

```
> fit.stl = stl(elecequip, t.window=13,
+               s.window="periodic", robust=TRUE)
>
> head(fit.stl$time.series, 2)
```

```
            seasonal    trend  remainder
Jan 1996 -4.951454 81.35054  2.9509180
Feb 1996 -5.776154 80.91799  0.6381671
```

```
> tail(fit.stl$time.series, 3)
```

```
            seasonal    trend remainder
Jan 2012 -4.951454 90.00316 1.3882959
Feb 2012 -5.776154 89.63097 1.1851810
Mar 2012  7.921218 89.25879 0.6199934
```

STL Example II

```
> fit.stl %>% seasadj() %>% naive()
```

```
         Point Forecast    Lo 80     Hi 80    Lo 95     Hi 95
Apr 2012       89.87878 85.24780  94.50976 82.79631  96.96125
```

```
> fit %>% forecast(method="naive")
```

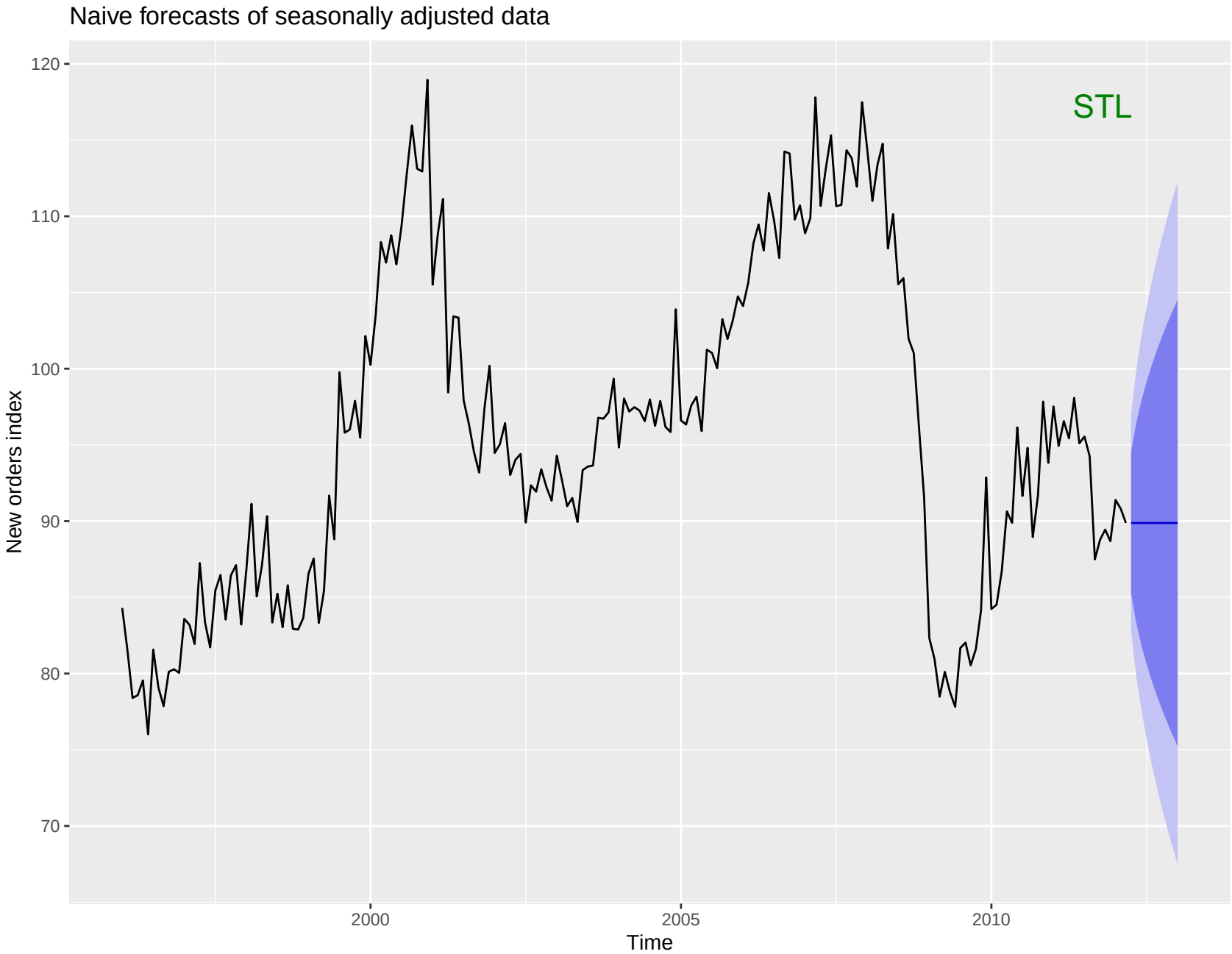
```
         Point Forecast    Lo 80     Hi 80    Lo 95     Hi 95
Apr 2012       83.89895 79.26798  88.52993 76.81648  90.98143
```

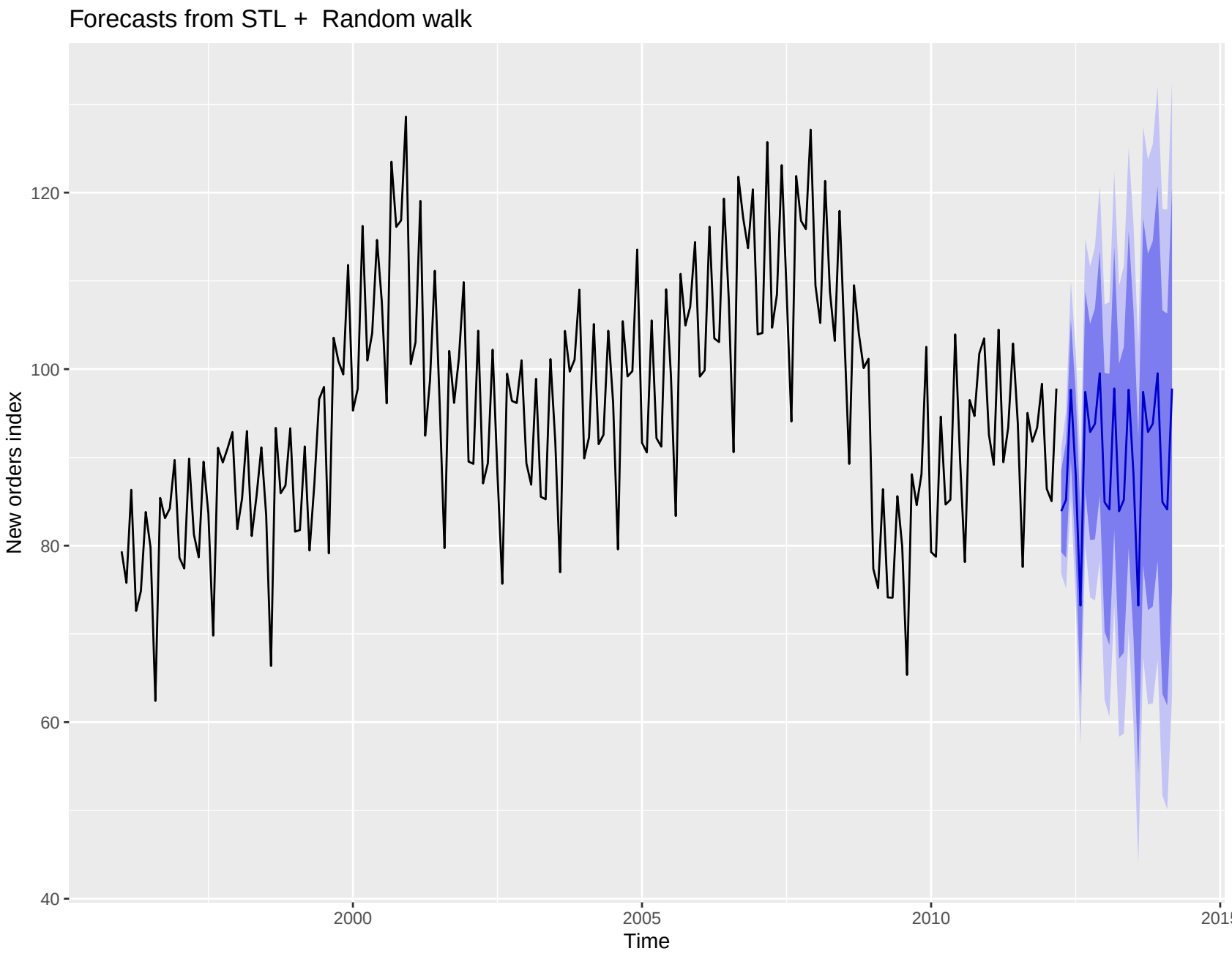
```
> tail(fit.stl$time.series, 12)
```

```
           seasonal    trend  remainder
Apr 2011  -5.979828 95.69042 -0.2605893
```

```
> 89.87878 - 5.979828
```

```
[1] 83.89895
```


Naive forecasts of seasonally adjusted data
STL
New orders index
120
110
100
90
80
70
2000
2005
2010
Time


Forecasts from STL + Random walk
New orders index
120
100
80
60
40
2000
2005
2010
2015
Time

- To consider more sophisticated models, you should know some extra notion.
- The mean function of a time series is defined to be

$$\mu(t) = \mathbb{E}[Y_t]$$

and the autocovariance function is defined to be

$$\gamma(s,t) = \text{cov}[Y_s, Y_t]$$

T different mean and variance + T(T-1)/2 number of covariance

- Notice there are $2T + T(T-1)/2$ parameters associated with a given time series $Y_1, \ldots Y_T$, which means it is not possible to estimate all of them.
- A key idea in time series is that of stationary, roughly speaking, a time series is stationary if its values always tend to vary about the same level and that their variability is constant over time.
- Stationary series have a rich theory and their behaviour is well understood.
- A time series with any trend-cycle, or seasonal component is not stationary!

- Formally, a time series is said to be strictly stationary if for any $k > 0$ and any $t_1, \ldots, t_k \in \mathbb{Z}$, the distribution of

$$(Y_{t_1}, Y_{t_2}, \ldots, Y_{t_k})$$

is the same as that for

garanteed that
variance is finite

$$(Y_{t_1+u}, Y_{t_2+u}, \ldots, Y_{t_k+u})$$

for every value of $u \in \mathbb{Z}$.

- It says that the behaviour of a stationary time series does not change over time, which means the following must be true if it is strictly stationary

$$\mu(t) = \mu(0)$$

and

$$\gamma(s,t) = \gamma(s-t,0)$$

- The two results of strictly stationary are often enough, people define a time series a weakly stationary if

$$\mu(t) = \mu(0)$$
$$\gamma(t+u, t) = \gamma(u, 0)$$
$$\mathbb{E}\left[Y_t\right]^2 < \infty \longrightarrow \text{variance is finite}$$

for all t and u.

- In the case of Gaussian time series, the two definitions are equivalent.
- When time series are stationary, it is possible to simply the parametrisation,

$$\mu(t) = \mu \qquad \text{and} \qquad \gamma(u) = \operatorname{cov}\left[Y_{t+u}, Y_t\right]$$

from which we can consider so-called, autocorrelation function

$$\rho(u) = \frac{\gamma(u)}{\gamma(0)}$$