Datorlaboration 5

Statistik och dataanalys 2 (ST1201), VT2023

Introduktion

Laborationen ska genomföras som en *Quarto-notebook*. En stor fördel med notebook-formatet är att det låter er skapa eget kursmaterial genom att kombinera kod med text som beskriver vad koden gör. Att skriva sitt egna kursmaterial och att med egna ord förklara hur saker fungerar är ett av dom bästa sätten att lära sig.

- Det är helt OK att ni samarbetar under labben, men skriv din egen labbrapport! Det är viktigt att faktiskt skriva koden själv.
- Om du fastnar, testa att se om du kan hitta en lösning i dokumentationen. För att se dokumentationen för en viss funktion skriver du ett frågetecken följt av funktionens namn i konsolen, exempelvis `?lm`. Funkar inte det, testa google eller chatGPT, och funkar inte det, fråga labbansvarig. Vi finns där för att svar på dina frågor, med det är viktigt att du tränar på att lösa problem själv.

Innehåll

I den här laboration kommer du lära dig att

- visualisera tidsseriedata,
- använda paketet `fpp3` för att göra två olika dekomponeringar, och
- undersöka AR(1)-processer med hjälp av en simuleringsstudie.

Innan du går vidare, skapa ett tomt quarto-dokument på samma sätt som du gjorde under första labben. Alltså, skapa ett nytt quarto-dokument, radera allt utom preamble, ändra `title` till något passande, och lägg sedan till en kodchunk (med rätt chunkinställningar) som du kan ha alla dina `library()`-anrop i och placera den precis efter preamble.

Del 1 - Visualisering av tidsseriedata

I den här uppgiften ska du jobba med ett subset av datasetet `aus_accommodation` från paketet `fpp3`. Du behöver alltså installer `fpp3` (om det inte redan är gjort) och lägga till `library(fpp3)` i din bibliotekschunk. Det första steget när vi jobbar med tidsserier är att se till att datasetet är kodad som en tidsserie. I detta fallet är `aus_accomodation` redan i `tsibble`-format, så du kan börja analysera det direkt.

- ☐ Använd `filter()` för att välja ut endast dom observationer som korresponderar till staten "Victoria". Spara som ett nytt dataset.
- ☐ Använd `mutate()` för att skap en ny variabel `Takings_adj` i ditt nya dataset genom att KPI-justera `Takings`. (Dvs, multiplicera `Takings` med 100 och dividera med `CPI`.)
- ☐ Använd `autoplot()` för att skapa en tidsserieplot av `Takings_adj`. Ser det ut att finnas en trend? Ser det ut att finnas en säsongskomponent?

`autoplot()` är en `ggplot2`-funktion som tittar på datasetet och avgör vilken figur som passar beroende på vilken sorts data den matas med. Datasetet du precis har skapat är en `tsibble` vilket betyder tidsseriedata, så `autoplot()` gör en tidsserieplot automatiskt! Det enda vi behöver ange i `autoplot()` är namnet på den variabel i vårt dataset vi vill plotta, i detta fall `Takings_adj`.

Det enklaste sättet att använda `autoplot()` är genom pipe-operatorn. För att skapa en tidsserieplot för variabeln `min_variabel` i datasetet `min_tidsserie` så kan du skriva

```
min_tidsserie %>%
  autoplot(min_variabel)
```

- ☐ Ofta är det rimligare att studera ekonomiska variabler på logskala. Använd `mutate()` för att skapa en ny variabel, log_takings_adj, genom att logaritmera `Takings_adj`. Under resten av uppgiften kommer du jobba vidare med log_takings_adj om inget annat nämns.
- ☐ Använd `autoplot()` för att skapa en tidsserieplot.

För att göra säsongsplottar kan du använda funktionen `gg_season`.

```
min_tidsserie %>%
  gg_season(min_variabel)
```

- ☐ Använd `gg_season()` för att skapa en säsongsplot. Under vilket kvartal är dom (logaritmerade) KPI-justerade intäkterna från turistnäringen (Takings) lägst?

Del 2 - Dekomponering

2a - Klassisk dekomponering

Följande kod genomför en klassisk dekomponering

```
min_tidsserie %>%
  model(classical_decomposition(min_variabel, type = "additive")) %>%
  components()
```

Du kan enkelt hoppa mellan "additiv" och "multiplikativ" dekomponering genom att ändra argumentet `type`. När vi väl har genomfört dekomponeringen kan vi sen använda en pipe för att skicka resultatet vidare till `autoplot()` så kommer den att ta fram diagram för trend, säsong, och rest!

- ☐ Utgå ifrån koden ovan och genomför en STL-dekomponering `log_takings_adj` från uppgift 1. Lägg till ytterligare en pipe följt av `autoplot()` för att plotta dom olika säsongskomponenterna.
- ☐ Förklara vad dom grå rektanglarna på vänster sida av figuren du precis skapade representerar.
- ☐ Testa med både additiv och multiplikativ modell. Varför blir den multplikativa så dålig?

2b - STL-dekomponering

Klassisk dekomponering har flera brister och används därför sällan. Ett något modernare alternativ är STL-dekomponering.

```
min_tidsserie %>%
  model(STL(min_variabel ~ trend() + season(), robust = TRUE)) %>%
  components() %>%
  autoplot()
```

- ☐ Modifiera koden ovan så att den gör en STL-dekomponering av `log_takings_adj` och plottar resultatet.
- ☐ `trend()` och `season()` tar båda argumentet `window =`, som avgör hur mycket data som ska användas när trend-cykel och säsongskomponenterna skattas. När vi lämnar `trend()` och `season()` tomma så väljer funktionen själv (ofta rimliga) värden på `window`. Testa först att ange `window = 1` och sen `window= 50` för trenden. Hur påverkas trendskattningen?

- [ ] Genom att ange `window = "periodic"` för `season()` så tvingar vi modellen att skatta säsongseffekten baserat på *samtliga* observationer. Utgå ifrån en modell med automatiskt vald trend, (så `trend()` utan något värde på `window` angivet) och testa vad som händer när du går ifrån en modell med automatiskt valt värde på `window` (`season()`) till en modell där säsongseffekten skattas baserat på samtliga observationer (`seasons(window = "periodic")`).

Del 3 - Autoregressiva processer

En autoregressiv process av ordning 1, AR(1), har följande populationsmodell

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t, \quad \varepsilon_t \overset{iid}{\sim} \mathrm{N}(0, \sigma^2_\varepsilon)$$

Om vi låter feltermen har varians 1 och att $y_0 = 0$, så kan vi *simulera* en tidsserie med koden nedan. Koden är en funktion som vi har skapat själva. Funktionen tar tre *argument*: `n`, `c`, och `phi1`, och *returnerar* sen en tidsserie. Tidsserien som skickas tillbaks kommer vara

- av längd `n`, alltså bestå av `n` observationer, och
- genererad från en AR(1) process med parametrarna `c` och `phi1`.

```
sim_AR1 <- function(n, c, phi1) {
  y <- rep(NA, n)
  y[1] <- c + phi1 * 0 + rnorm(1)
  for (i in 2:n) {
    y[i] <- c + phi1 * y[i - 1] + rnorm(1)
  }
  dfy <- tsibble(
    y = y,
    t = 1:n,
    index = t
  )
  return(dfy)
}
```

Ett sätt att se detta på är att vi genererar en *realisation* av en tidsserie från en given populationsmodell. För att undersöka hur en tidsserie av längd `n = 100` vars populationsmodell ges av

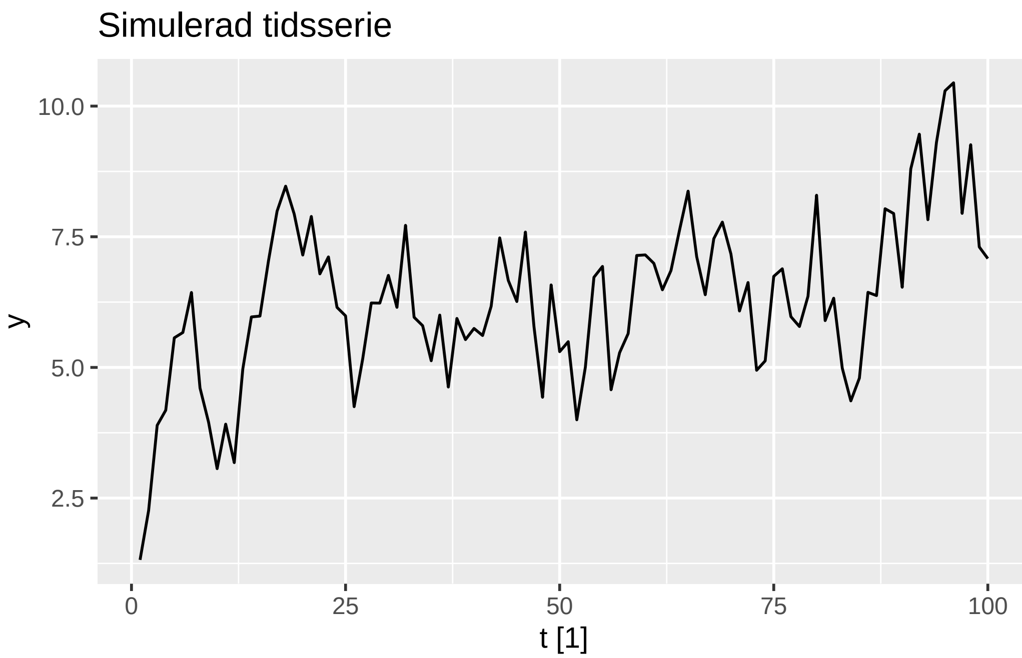
$$y_t = 2 + 0.7 y_{t-1} + \varepsilon, \quad \varepsilon_t \overset{iid}{\sim} \mathrm{N}(0, 1)$$

kan se ut så kan vi helt enkelt testa att generera en tidsserie med

- `c = 2`
- `phi1 = 0.7`

- `n = 100`

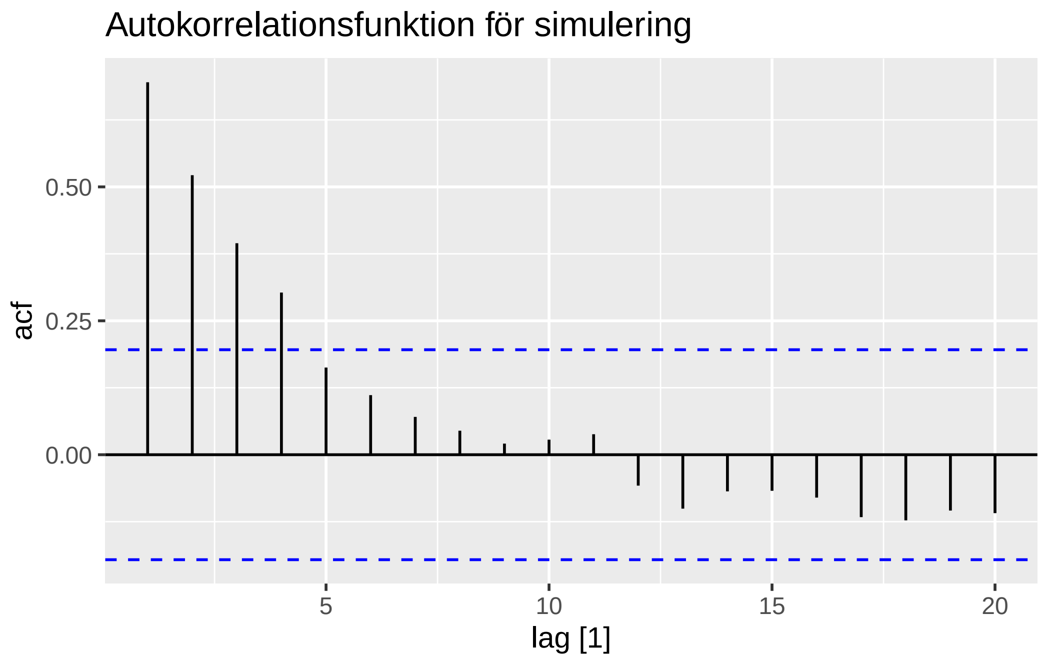
```
sim_tidsserie <- sim_AR1(n = 100, c = 2, phi1 = 0.7)
sim_tidsserie %>%
  autoplot(y) +
  labs(title = "Simulerad tidsserie")
```



Det vi gör nu är en typ av *simuleringsstudie*. Vi utgår ifrån en specifik populationsmodell, och sen undersöker vi hur olika faktiska tidsserier som skapats med hjälp av den populationsmodellen kan komma att se ut.

En annan vanlig sak vi vill göra med tidsseriedata är att undersöka (sample-)autokorrelationsfunktionen. Vi kan göra detta med funktionen `ACF()` tillsammans med `autoplot()`.

```
sim_tidsserie %>%
  ACF(y, lag_max = 20) %>%
  autoplot() +
  labs(title = "Autokorrelationsfunktion för simulering")
```



- Om du blir förvånad över hur autokorrelationen ser ut kan du testa att ändra `n` till 10000!

Ditt jobb nu är att skapa realisationer av olika AR(1)-processer, och sen analysera dom. Om du vill se till att dina realisationer inte ändrar sig varje gång du renderar din notebook så kan du sätta ett slumpfrö (`set.seed`)! Detta är ofta en bra idé när man jobbar med slumpmässiga data.

3a - Vitt brus

Om vi sätter `phi1 = 0` så får vi följande populationsmodell

$$y_t = c + \varepsilon, \quad \varepsilon_t \overset{iid}{\sim} \mathrm{N}(0, 1)$$

Alltså, vi får en process där värdet vid tidpunkten t inte beror alls på värdet vid tidpunkten $t-1$. Denna typ av process kallas för *vitt brus*.

- ☐ Simulera fram en tidsserie av längd `n = 20` med vitt brus. Du kan välja `c` precis som du vill.
- ☐ Plotta tidsserien med `autoplot()`.
- ☐ Ta fram en figur som visar autokorrelationsfunktionen för tidsserien.
- ☐ Gör om dom tre föregående stegen, men använd `n = 1000` istället. Tankar?

3b - Stationära AR(1)-processer

Alla AR(1)-processer med `phi1` mellan -1 och 1 är *stationära*. Detta betyder ungefär att dom är stabila över tid. Dom kan "vandra omkring", men har ett medelvärde som kommer hålla sig i närheten av.

- ☐ Simulera fram en tidsserie av längd `n = 1000` med valfritt värde på `c` och med `phi1` = 0.5.
- ☐ Plotta tidsserien med `autoplot()` och skapa en figur som visar autokorrelationsfunktionen.
- ☐ Återupprepa dom två föregående stegen, men använd nu `phi1 = -0.5`.

3c - Slumpvandring

Ett klassiskt exempel på en icke-stationär tidsserie är en *slumpvandring* (random walk på engelska). Om vi sätter $c = 0$ och $\phi_1 = 1$ i populationsmodellen får vi en slumpvandring

$$y_t = y_{t-1} + \varepsilon, \quad \varepsilon_t \overset{iid}{\sim} \mathrm{N}(0, 1)$$

En slumpvandring är en vild process som kan bete sig på lite olika sätt. Bland det roligaste man kan göra som statistiker är att simulera fram olika slumpvandringar eftersom att det *nästan alltid* ser ut som att det finns tydliga mönster även när det inte borde vara så!

- ☐ Simulera fram tre slumpvandringar av längd 1000 och plotta med `autoplot()`.