

R-programmering VT2022

Föreläsning 6

Johan Alenlöv

2022-02-28

Linköpings Universitet

Föreläsning 6

Innehåll föreläsning 6

- Information
- Programmeringsparadigm
 - Funktionell programmering
 - Objektorientering
- Datum och tid med `lubridate`
- Linjär algebra i R

Information

- Kom ihåg att anmäla er till datortentan
 - Endast anmälda får skriva tentan
- Datortentan kommer att vara på plats
 - Tenta i SU-sal
 - Testsession, 22/3 10:15-12:00 i SU-17/18
- Komplettering labb 1-4: första tillfället samtidigt som inlämning för labb 6.
 - Se kurshemsidan för information

Programmeringsparadigm

- Finns många olika sätt att se på **hur** program ska skrivas.
- Olika programmeringsspråk följer olika paradigmer.
- Pratar ofta om två för R:
 - Funktionell programmering
 - Objektorienterad programmering

- I funktionell programmering pratar man om två saker.
 - “Rena” funktioner.
 - Output beror bara på inargumenten.
 - Funktionen har inga sidoeffekter.
 - “First-class functions”.
 - Funktioner ska bete sig som alla andra objekt
 - Kunna spara funktioner, returnera funktioner etc.
- Exempel:
 - Scala, Erlang, Haskell

- Objektorientering åtgår från klasser:
 - En klass är en mall som används för att skapa objekt
 - Kopplar samman objekt med funktioner
 - Objekt är en specifik realisering av en klass
 - Objekt skapas med en konstruktor
- Viktiga koncept:
 - **Inkapsling**: Gömma och ordna relaterad data.
 - **Polymorfism**: Generiska funktioner som hanterar olika objekt.
 - **Arv**: Underlättar specialisering av data och metoder via underklasser.

- i R:
 - Alla objekt ha en klass
 - Kan undersöka ett objekts klass med `class()`
 - Variabeltyper är “atomära” klasser
 - Klasser har en konstruktor
- Olika typer av klasser i R:
 - Basklasser
 - S3 (informellt): “lista med klassattribut”
 - S4 (formellt): element väljs ut med `@`
 - Reference classes

Generiska funktioner (S3)

- Objektorientering i R utgår från generiska funktioner
- Funktioner som gör olika saker beroende på objektets klass
- Ex:
 - `plot()`
 - `mean()`
 - `summary()`
 - `names()`
- Funktionerna anropar metoden för objektet när de används.

- En funktion för en specifik klass
- Metoderna utgår från generisk funktion
- Alla metoder för en generisk funktion hittas med `methods()`
- T.ex. funktionen `mean()`
 - Defaultmetoden finns med `mean.default()`
 - Specifik med `mean.difftime()`
- Metoder kan också anropas direkt som vanliga funktioner.

Datum och tid

- Datum är klurigt att arbeta med, men används extremt mycket.
- Två typer av tid:
 - Relativ tid
 - Exakt tid
- Enklare funktioner för datum finns i base

Datum och tid - Exempel

```
my_date <- "2022-02-28"  
class(my_date)
```

```
## [1] "character"
```

```
my_date_as_date <- as.Date(my_date)  
my_date_as_date
```

```
## [1] "2022-02-28"
```

```
class(my_date_as_date)
```

```
## [1] "Date"
```

- Paket för enkel och effektiv datumhantering
- Sammansatt av “lubricant” och “date”

Tre huvudsakliga delar:

1. Läs in datum
2. Ändra inlästa datum
3. Göra beräkningar med datum

lubridate - Exempel

```
library(lubridate)
idag <- ymd("2022-02-28")
print(idag)
```

```
## [1] "2022-02-28"
```

```
week(idag)
```

```
## [1] 9
```

```
idag + weeks(2)
```

```
## [1] "2022-03-14"
```

Elementordning	Funktion
år, månad, dag	<code>ymd()</code>
år, dag, månad	<code>ydm()</code>
månad, dag, år	<code>mdy()</code>
timme, minut	<code>hm()</code>
timme, minut, sekund	<code>hms()</code>
år, mån, dag, timme, min, sek	<code>ymd_hms()</code>

Källa: Grolemund and Wickham (2011, Table 4)

För att “plocka ut” eller ändra delar av ett datum används följande funktioner

Datum	Funktion	Tidsdel	Funktion
år	<code>year()</code>	timme	<code>hour()</code>
månad	<code>month()</code>	minut	<code>minute()</code>
vecka	<code>week()</code>	sekund	<code>second()</code>
årsdag	<code>yday()</code>	tidszon	<code>tz()</code>
månadsdag	<code>mday()</code>		
veckodag	<code>wday()</code>		

Källa: Grolemund and Wickham (2011, Table 5)

lubridate - Exempel

```
idag <- ymd("2022-02-28")  
week(idag)
```

```
## [1] 9
```

```
wday(idag, label = TRUE)
```

```
## [1] Mån
```

```
## Levels: Sön < Mån < Tis < Ons < Tor < Fre < Lör
```

```
year(idag) <- 2019  
idag
```

```
## [1] "2019-02-28"
```

- För att räkna med datum finns det fyra olika objekt i

lubridate

- instant
- interval
- duration
- period

Instant

- Ett specifikt tillfälle i tiden
- Viktiga funktioner:
 - `now()`
 - `today()`

Interval

- Tidsspannet mellan två `instant`
- `interval(start,end)`

instant och interval - Exempel

```
inst_1 <- ymd("2020-02-29")
inst_2 <- today()
my_interval <- interval(start = inst_1, end = inst_2)
my_interval
```

```
## [1] 2020-02-29 UTC--2022-02-28 UTC
```

- Ett fixt tidsspann som mäts i sekunder
- Tänk kontinuerlig tid
- Absolut tid i sekunder
- Funktioner börjar med d
- Konvertera ett interval med `as.duration()`
- Ex:
 - `duration()`
 - `dseconds()`
 - `dhours()`

duration - Exempel

```
my_interval / ddays(30)
```

```
## [1] 24.33333
```

```
as.duration(my_interval)
```

```
## [1] "63072000s (~2 years)"
```

- Utgår från den aktuella enheten (dagar, månader, år)
- Tänk diskret tid
- Relativ tid
- Vad vi i dagligt tal menar med ex. två veckor
- Ex:
 - `period(num = , units =)`
 - `minutes()`
 - `hours()`
 - `weeks()`

Period - Exempel

```
months(3) + weeks(2) + days(1)
```

```
## [1] "3m 15d 0H 0M 0S"
```

```
my_interval %/% months(4)
```

```
## [1] 6
```

```
as.period(my_interval)
```

```
## [1] "1y 11m 30d 0H 0M 0S"
```

Att räknamed tid - Exempel

```
course_start <- ymd("2022-01-24")  
course_start + weeks(5)
```

```
## [1] "2022-02-28"
```

```
course_start + months(1)
```

```
## [1] "2022-02-24"
```

```
course_start + ddays(30)
```

```
## [1] "2022-02-23"
```

Linjär algebra

- Matriser är två-dimensionella vektorer
- Har jobbat med matriser förut.
- Skapa en matris med
 - `matrix(data = , nrow = , ncol = , byrow =)`
 - Kom ihåg cirkulering
 - `byrow = FALSE` per default
- Transponat fås med `t()`
- Diagonalmatris med `diag(vektor)`
- Enhetsmatrisen med `diag(nummer)`

- Matrimultiplikation görs med `%*%`
- Invertera en matris med `solve()`
 - Notera att `A %*% solve(A)` blir en enhetsmatris
- Lösa ekvationssystem $Ax = b$ med `solve(a=A,b=b)`
 - Notera att om A inverterbar är svaret $x = A^{-1}b$.
- Egenvärden och egenvektorer med `eigen()`
 - Returnerar en lista med egenvärden och egenvektorer
- Summera rader eller kolumner med `rowSums()` och `colSums()`
- Kombinera matriser med `rbind(,)` eller `cbind(,)`

- **Matrix** är ett paket som implementerar effektivare/snabbare funktioner för linjär algebra.
- Grunden är funktionen `Matrix()`, vilket är en konstruktör som skapar matriser.
- Matriserna i **Matrix** är S4 klasser.
- I **Matrix** är det skillnad på glesa och täta matriser.
- Kolla dokumentation för mer information.
- Rekommenderas för avancerade tillämpningar.