Inlämning 2

Josef Wilzén

19 april 2025

Inlämning

Utgå från laborationsmallen när du gör inlämningsuppgifterna. Det finns en mall för varje inlämningsuppgift, se nedan under Inlämningsuppgifter. Filen ni lämnar in ska **inte** innehålla något annat än de aktuella funktionerna, namn- och ID-variabler och ev. kommentarer.

För att lämna in ska ni gör följande:

- Lös uppgifterna enligt beskrivningarna i instruktionen.
- Se till att ni kan återskapa de testfall som visas i uppgiften.
- Se till att era funktioner klara alla tester i `markmyassignment`. Det rekommenderas att använda `markmyassignment` först när er funktion är hyfsat klar.
- På en av de schemalagda datorlaborationerna: kontakta en lärare/assistent och be om att få redovisa er inlämning.
- **Redovisning:**
  - Rensa den globala miljön (Environment i Rstudio) så att bara de variabler och funktioner som krävs för uppgiften finns kvar. Tips: Funktionen `rm()` tar bort objekt/variabler
  - Kör markmyassignment och visa att testerna går igenom.
  - Förklara kort hur ni har löst uppgifterna och svara på eventuella frågor från assistenten.
  - Om assistenten säger att ni är godkända, lämna då in er lösning på Inlämningar i kursrummet på Lisam.
- **Deadline** finns under Inlämningar i kursrummet på Lisam.

Tips!

Inlämningsuppgifterna innebär att konstruera funktioner. Ofta är det bra att bryta ned programmeringsuppgifter i färre små steg och testa att det fungerar i varje steg.

1. Lös uppgiften med vanlig kod direkt i RStudio (precis som i datorlaborationen ovan) utan att skapa en funktion.
2. Testa att du får samma resultat som testexemplen.
3. Implementera koden du skrivit i 1. ovan som en funktion.
4. Testa att du får samma resultat som i testexemplen, nu med funktionen.

Automatisk återkoppling med `markmyassignment`

Som ett komplement för att snabbt kunna få återkoppling på de olika arbetsuppgifterna finns paketet `markmyassignment`. Med detta är det möjligt att direkt få återkoppling på uppgifterna i laborationen, oavsett dator. Dock krävs internetanslutning.

Information om hur du installerar och använder `markmyassignment` för att få direkt återkoppling på dina laborationer finns att tillgå här.

Samma information finns också i R och går att läsa genom att först installera `markmyassignment`.

```
install.packages("markmyassignment")
```

Om du ska installera ett paket i PC-pularna så behöver du ange följande:

```
install.packages("markmyassignment",lib="mapp i din hemkatalog")
```

Tänk på att i sökvägar till mappar/filer i R i Windowssystem så används `"\\"`, tex `"C:\\Users\\Josef"`.

Därefter går det att läsa information om hur du använder `markmyassignment` med följande kommando i R:

```
vignette("markmyassignment")
```

Det går även att komma åt vignetten här. Till sist går det att komma åt hjälpfilerna och dokumentationen i `markmyassignment` på följande sätt:

```
help(package="markmyassignment")
```

Lycka till!

Kapitel 1

Inlämningsuppgifter

Ladda nu ner laborationsmallen, som finns **här** och spara namn och liuID i respektive variabel. Spara filen som `inl2_[ditt liuID].R` Här är ett exempel `inl2_joswi123.R` .

För att använda `markmyassignment` i denna laboration ange:

```
library(markmyassignment)
# eller
library(markmyassignment,lib="en mapp i din hemkatalog")
lab_path <-
"https://raw.githubusercontent.com/STIMALiU/KursRprgm2/main/Labs/Tests/inl2.yml"
set_assignment(lab_path)

Assignment set:
Inl2: Statistisk programmering med R: Inlämning 2
The assignment contain the following (2) tasks:
- blood_match
- hilbert_matrix
```

1.1 Lösa inlämningsuppgifter

Här kommer lite tips till när ni ska lösa inlämningsuppgifter.

- Inlämningsuppgifter utgår ifrån att ni har gjort **övningsuppgifterna** ovan först. Tänk att ni ska göra minst 70 % av övningsuppgifterna innan ni börjar med inlämningsuppgifterna.
- Se först till att ni förstår ”problemet” i uppgiften. Vad ska funktionen göra? Beskriv problemet för er själva. Vissa problem är det bra att bryta ner i mindre delproblem.
- Börja med att lösa problemet (eller det första delproblemet) med vanlig kod (dvs inte i en funktion) i ett R-script.
  - Lös ett delproblem i taget vid behov. Sätt sedan samman lösningarna på delproblemen. Under sök om koden fungerar som den ska.
- Sätt sedan in er kod i en funktion. Testa om funktionen kan återskapa de exempel som visas under beskrivningarna (testfallen). Se till att funktionen har rätt namn och rätt namn på argumenten.
- Ta er funktion och lägg in den i kodmallen, dvs i ett nytt R-script. Skapa variablerna `Namn` och `LiuId` med rätt innehåll. Spara filen med rätt namn.
- Kommentera bort all kod i er fil som inte är de aktuella funktionerna, namn- och ID-variabler och ev. kommentarer.
  - Detta minskar risken för fel.
- Testa nu att använda markmyassignment för att rätta er funktion.

  - Använd markmyassignment bara när er funktionen är ”hyfsat klar”, annars kommer ni att få många fel i markmyassignment, och det är svårt att veta var man ska börja kolla.
  - Det är vikitgt att funktionen rätt namn och rätt namn på argumenten, annars kommer ni att få många fel i markmyassignment.
- Om ni inte klarar alla tester i markmyassignment: undersök vilka felmeddelanden som ni får. Ta hjälp av lärare/labbassitenter vid behov om det är svårt att tolka.
- Innan ni lämnar in:
  - Se till att ni har löst uppgiften på det sätt som den är beskriven i uppgiftstexten. Ibland måste en viss metod användas för att lösa uppgiften. Ibland är vissa funktioner inte tillåtna i en viss uppgift. Så läs noga i varje uppgift vad som gäller.
  - Se till att ni klarar alla tester i markmyassignment

Dokumentation och kodstil

Från och med denna laboration och de resterade laborationerna i kursen så ska ni förutom att lösa angivna uppgifter också **kommentera** era funktioner och ha en **god kodstil** för att bli godkända.

- Kodstil:
  - Det viktiga är att koden ska vara **tydlig** och **läsbar**.
  - Följ någon av kodstilarna i Datorlaboration 4. Ni måste inte följa dessa exakt, men koden ska se bra ut och var konsekventa i den stil som ni väljer att använda.
  - Tänk särskilt på:
    * Enhetlighet och struktur
    * Ha lämplig indentering och avstånd
    * Ha bra variabelnamn
- Kommentarer:
  - Funktionshuvud: Använd mallen för `roxygen2` som ges i Datorlaboration 4. Ni ska ha med:
    * `@title` Här skriver ni funktionens namn
    * `@description` Förklara kort vad funktionen gör
    * `@param` Skriv först arguments namn, sen mellanslag och sen förklara kort argumentet. Upprepa detta för alla argument i funktionen. Ex: `x Numerisk vektor, används vid beräkning av medelvärde.`
    * `@return` Förklara vad funktionen returnerar
  - Kommenter i funktionen: Era lösningar ska innehålla lämpliga kommenater, där ni förklarar de övergripande dragen och de viktiga stegen i er kod. Ni behöver inte föklara alla detaljer. Kommentarerna ska berätta sådant för programmeraren som inte står i koden. Använd luft och kommentarer för att gruppera och strukturera er kod.
  - Tips: Tänk att det ska vara lätt att förstå er funktion långt senare, tex om ett år. Vilka kommentarer behövs då?

För att bli godkänd på inlämningsuppgifterna måste ni följa ovanstående instruktioner för kommentarer och kodstil.

1.2 `blood_match()`

Vid en blodtransfusion är det viktigt att hålla koll på givarens och mottagarens blodgrupp. Blodgrupperna delas in i AB0 och Rh systemet.

För AB0 systemet gäller att en person kan antingen vara typ A, B, AB eller 0. Följande gäller:

- En person med typ AB kan ta emot blod från alla typer
- En person med typ A kan bara ta emot från någon med typ A eller 0

- En person med typ B kan bara ta emot från någon med typ B eller 0
- En person med typ 0 kan bara ta emot från en person med typ 0.

För Rh systemet gäller att en person kan antingen vara positiv (+) eller negativ (-). Följande gäller:

- En person som är positiv kan ta emot blod från båda typerna.
- En person som är negativ kan bara ta emot blod från någon som är negativ.

Er uppgift är att skapa en funktion `blood_match()` med argumentet `patients` som talar om ifall en person kan ta emot en annan persons blod baserat på deras blodgrupp. Ni får givet en lista `patients` som ska innehålla två listor som heter `giver` och `receiver`, en för varje person. Varje person har två variabler, som heter `ABO` och `rh` som båda innehåller en textsträng, där textsträngarna ska vara A,B,AB eller 0 för `ABO` och + eller - för `rh`. Ett exempel på en sådan lista visas nedan:

```
# exempel på lista som ges som input till funktionen:
test_patientsA = list(giver = list(ABO = "O", rh = "-" ),
        receiver = list(ABO = "AB", rh = "+"))
str(test_patientsA)

List of 2
 $ giver   :List of 2
  ..$ ABO: chr "O"
  ..$ rh : chr "-"
 $ receiver:List of 2
  ..$ ABO: chr "AB"
  ..$ rh : chr "+"
```

Funktionen ska skriva ut textraden `They are a match` om transfusionen är möjlig. Om det inte är möjligt ska texten `They are not a match` skrivas ut tillsammans med `Incompatible ABO`, `Incompatible rh` eller `Incompatible ABO and rh` beroende på vad som inte matchade. Funktionen ska alltså använda `print()` för att ge denna output, ni ska inte använda `return()` i denna funktion.

Om det är så att patients **inte** är en lista ska funktionen **stoppas** och returnera felmeddelandet **Not a list**

Om det är så att någon av värdena i AB0 och rh **inte** är korrekta (t.ex. AB0 skulle vara C) ska funktionen **stoppas** och returnera felmeddelandet **Wrong bloodtype**

Exempel på hur man kan implementera är:

1. Testa så att argumentet patients är en lista.
2. Kolla så att AB0 och rh-värdena är tillåtna. *tips: skapa en vektor med tillåtna värden och använda %in% för att se om det givna värdet finns i denna vektor*
3. Skriv en if - else if - else del för att testa om AB0 värdena stämmer.
4. Skriv en if - else if -else del för att testa om rh värdena stämmer.
5. Kombinera resultaten från punkt 3 och 4 för att få rätt utskrift.

Här kommer ett exempel på hur funktionen ska fungera:

```
test_patientsA = list(giver = list(ABO = "O", rh = "-" ),
        receiver = list(ABO = "AB", rh = "+"))
blood_match(test_patientsA)

[1] "They are a match"

test_patientsB = list(giver = list(ABO = "AB", rh = "+" ),
        receiver = list(ABO = "O", rh = "+"))
blood_match(test_patientsB)
```

```
[1] "They are not a match, Incompatible ABO"

test_patientsC = list(giver = list(ABO = "O", rh = "+" ),
        receiver = list(ABO = "B", rh = "-"))
blood_match(test_patientsC)

[1] "They are not a match, Incompatible rh"

test_patientsD = list(giver = list(ABO = "A", rh = "H" ),
        receiver = list(ABO = "AB", rh = "+"))
blood_match(test_patientsD)

Error in blood_match(test_patientsD): Wrong bloodtype

# Om ingen lista
blood_match("hej!")

Error in blood_match("hej!"): Not a list
```

1.3 `hilbert_matrix()`

Vi ska nu skapa en funktion för att skapa godtyckligt stora Hilbertmatriser. Varje element i en Hilbertmatris är definierat som

$$H_{ij} = \frac{1}{i+j-1} \tag{1.1}$$

där i är radindex och j är kolumnindex. För mer information om Hilbertmatriser finns **här**. Skapa en funktion `hilbert_matrix()` med argumenten `nrow` och `ncol`. Funktionen ska returnera en Hilbertmatris enligt ovan. I denna uppgift är det praktiskt att använda en nästlad for-loop. Exempel på implementering:

1. Skapa en matris H av rätt storlek
2. Skapa en nästlad for-loop, där en loop går över rader och en över kolumner i H. Se till att ha olika namn på de två loopvariablerna.
3. Använd formel 1.1 för att beräkna varje element i matrisen H med hjälp av loop-index för de två looparna.
4. Returnera H

I denna uppgift är det inte tillåtet att använda sig av inbyggda funktioner eller R-paket som direkt beräknar Hilbertmatriser, utan ni måste skapa funktionen själva enligt instruktionerna ovan. Exempel på otillåtna funktioner är: `Hilbert()` i `Matrix` och `pbdDMAT`, `hilbert.matrix()` i `matrixcalc`, `hilb()` i `pracma`.

Nedan finns exempel på hur funktionen ska fungera.

```
hilbert_matrix(nrow = 1, ncol = 4)

     [,1] [,2]    [,3] [,4]
[1,]    1  0.5 0.33333 0.25

hilbert_matrix(nrow = 3, ncol = 1)

        [,1]
[1,] 1.00000
[2,] 0.50000
[3,] 0.33333

hilbert_matrix(nrow = 5, ncol = 2)
```

```
        [,1]    [,2]
[1,] 1.00000 0.50000
[2,] 0.50000 0.33333
[3,] 0.33333 0.25000
[4,] 0.25000 0.20000
[5,] 0.20000 0.16667
```

```
A<-hilbert_matrix(nrow = 4, ncol = 4)
A
```

```
        [,1]    [,2]    [,3]    [,4]
[1,] 1.00000 0.50000 0.33333 0.25000
[2,] 0.50000 0.33333 0.25000 0.20000
[3,] 0.33333 0.25000 0.20000 0.16667
[4,] 0.25000 0.20000 0.16667 0.14286
```

```
hilbert_matrix(3, 3)
```

```
        [,1]    [,2]    [,3]
[1,] 1.00000 0.50000 0.33333
[2,] 0.50000 0.33333 0.25000
[3,] 0.33333 0.25000 0.20000
```

```
hilbert_matrix(2, 3)
```

```
     [,1]    [,2]    [,3]
[1,]  1.0 0.50000 0.33333
[2,]  0.5 0.33333 0.25000
```

```
B<-hilbert_matrix(nrow = 4, ncol = 6)
str(B)
```

```
 num [1:4, 1:6] 1 0.5 0.333 0.25 0.5 ...
```

```
print(B)
```

```
        [,1]    [,2]    [,3]    [,4]    [,5]    [,6]
[1,] 1.00000 0.50000 0.33333 0.25000 0.20000 0.16667
[2,] 0.50000 0.33333 0.25000 0.20000 0.16667 0.14286
[3,] 0.33333 0.25000 0.20000 0.16667 0.14286 0.12500
[4,] 0.25000 0.20000 0.16667 0.14286 0.12500 0.11111
```

Tänk på att filen du lämnar in endast ska innehålla de obligatoriska variablerna och funktionerna i inlämningsuppgifterna och **inget** annat. Var noga med att ge filen korrekt namn innan du lämnar in den.

Grattis! Nu är du klar!