Datorlaboration 4

Måns Magnusson

January 17, 2014

Instruktioner

- Denna laboration ska göras **en och en**.
- Det är tillåtet att diskutera med andra, men att plagiera eller skriva kod åt varandra är **inte tillåtet**.
- Utgå från laborationsfilen som går att ladda ned **här**
- Laborationen består av två delar:
  - Datorlaborationen
  - Inlämningsuppgifter
- Innan du lämnar in laborationen:
  1. Starta om R-Studio eller rensa den globala miljön (Global environment) med `rm(list = ls())`.
  2. Ladda in funktionerna i R med `source`.
  3. Kontrollera att inget annat än funktionerna laddas in.
  4. Testa att funktionerna fungerar en sista gång.
- Deadline för labben framgår på **kurshemsidan**

Contents

Part I
Datorlaboration

1 Mer om funktioner

1. Skapa följande funktion i R med både {} och `return()`:

$$f(x, y) = x^2 + y^2 + z^2 - 1$$

2. Vilken är den fria variabeln? Studera funktionen på följande sätt:

```
f
```

3. Skapa funktionen igen men gör det med en enda rad kod utan {} eller `return()`.

4. Pröva lite olika värden på `x` och `y`. Pröva att använda argumentnamnen och byt ordning på `x` och `y` i funktionsanropet. Eftersom `z` inte är definierad, kommer funktionen inte fungera om du inte sätter `z` till något värde. Pröva funktionen både före och efter du satt `z` till ett godtyckligt värde.

```
f
```

5. Pröva nu att beräkna följande integral. Börja med att definiera funktionen och använd sedan funktionen `integrate()`.

$$\int_0^3 \frac{1}{3}x^2 dx$$

6. Pröva nu att integrera en exponentialfördelnings täthetsfunktion med $E(X) = 1$ mellan 0 och 1:

$$\int_0^1 f_X(x)dx$$

D.v.s. vilken sannolikhetsmassa finns mellan 0 och en exponentialfördelnings medelvärde. **Här** finns exponentialfördelningens täthetsfunktion. **Obs!** $E(X) = \frac{1}{\lambda}$

7. Gör om uppgift oven, men använd lägg till argumentet `lambda` i funktionen som integreras. Sätt argumentet `lambda` till 1 som standard. Prova om du får samma resultat som ovan.

8. Pröva att använda din nya genom att numeriskt integrera följande funktion där $f_X(x)$ är en exponentialfunktion med $E(X) = 5$. I funktionen `integrate()` finns `...` för godtyckliga argument som ska skickas till din funktion. Pröva att på så sätt ändra värdet på `lambda` när du integrerar numeriskt.

$$\int_0^5 f_X(x)dx$$

1.1 Globala och lokala miljöer i R

1. Skapa följande funktion och pröva denna kod:

```
c <- function(x, y) {
    x + y
}
```

2. Den funktion vi brukar använda för att skapa vektorer kallas också `c()`. Vilken funktion är det som används om du anropar funktionen `c` nu?

3. R följer en given sökväg för när funktionen `c` anropas. Studera hur sökvägen ser ut med search(). För att titta i vilken namespace den funktion du just skapat ligger pröva funktionen `environment()`. För att ta reda på var den gamla funktionen (för att skapa en vektor) ligger, titta i hjälpen efter `c`. Högst upp till vänster ska det stå i vilken namespace funktionen ligger. Är den namespace där din skapade funktion ligger tidigare i R:s sökväg än den gamla `c`-funktionen?

4. Pröva nu att anropa den gamla c-funktionen med hjälp av `::` direkt från den namespace den gamla funktionen `c` ligger i.

5. Ta nu bort din `c` - funktion med `rm(c)`.

6. Skapa nu följande funktionen nedan. Vilken variabel är en så kallad fri variabel?

```
f <- function(x) {
    (x + y)^2 - 1
}
```

7. Pröva först att använda funktionen utan att ge `y` ett värde (eller om den finns i global enviroment, ta bort den). Fungerar funktionen `f`? Pröva att ge variabeln `y` ett värde. Fungerar funktionen nu?

8. R söker efter värden på fria variabler på samma sätt som R söker efter funktioner när ett funktionsnamn anropas. Dock påbörjar sökningen efter fria variabler i den miljö där funktionen är definierad. Skapa koden nedan, vad gör funktionen?

```
y <- 5
g <- function(x) {
    y <- 10
    # f here
    print(environment(f))
    output <- f(x)
    return(output)
}
```

9. Pröva att istället definiera funktionen `f` inne i funktionen som markerats med en kommentar. Vad blir skillnaden om du nu kör `g()`?

10. Prova att använda funktionen `assign()` för att göra följande operation:

```
svar <- 42
```

11. Pröva att med `assign()` skapa variablerna `svar1`, `svar2`, `svar3`, ..., `svar10` med en `for` loop. Varje svar (1 till 10) ska vara 42.

12. Vi ska nu skapa en funktion du kallar `minianalys()` som tar en godtycklig numerisk vektor och skriver ut vektorns medelvärde, standardavvikelse och kvantiler till skärmen. Pröva funktionen på vektorerna nedan: [**Tips!** `quantile()`]

```
utanNA <- 1:100
medNA <- c(rep(NA, 10), 11:100)
```

13. Samtliga de funktioner inne i `minianalys()` kan ta argumentet na.rm för att rensa bort NA innan beräkningar görs. Använd ellipsis-argumentet `...` för att möjliggöra att "skicka vidare" argumentet `na.rm` till funktionerna inne i `minianalys()`. Pröva sedan vektorerna ovan men med argumentet `na.rm = TRUE` i `minianalys()`.

2 `*apply` funktioner

1. Vi ska nu använda så kallade de så kallade `*apply`-funktionerna i R. Dessa funktioner används istället för loopar. De är ofta betydligt mycket snabbare att beräkna än loopar.
2. Vi börjar med funktionen `tapply()`. `tapply()` används för att använda en funktion per grupp (över en så kallad "Ragged array" eller vektorer av olika längd). Detta är ofta av intresse i praktiken. Vi börjar med att läsa in datsetet `ChickWeight`.

```
data(ChickWeight)
```

3. Vi ska nu pröva `tapply()` som har argumenten `X`, `INDEX`, `FUN` och `simplify`. `X` anger variabeln (eller datasetet) vi vill använda funktionen på, `INDEX` anger vilken gruppvariabel som ska användas och FUN anger vilken funktion som ska användas per grupp. Ett exempel på hur vi kan beräkna den genomsnittliga vikten per kyckling ser ut på följande sätt:

```
tapply(X = ChickWeight$weight, INDEX = ChickWeight$Chick, FUN = mean)
```

4. Pröva att på ett liknande sätt beräkna standardavvikelsen för varje kyckling samt antalet observationer (längden av vektorn) och kvantilerna med `quantile()`.
5. Pröva nu att skicka argument till quantile för att räkna ut percentiler för varje kyckling.
6. `*apply`-funktioner är särskilt smidiga att använda tillsammans med anonyma funktioner. Pröva koden nedan. Vad gör den?

```
tapply(X = ChickWeight$weight, INDEX = ChickWeight$Chick, FUN = function(x) x)
```

7. Pröva att på liknande sätt skapa en funktion som räknar ut skillnaden mellan det första värdet och det sista värdet för varje kyckling med en anonym funktion.
8. Pröva nu att göra om uppgiften ovan, men i `tapply()` ange `simplify = TRUE`. Vad är skillnaden?
9. Vi ska nu studera `lapply()`. `lapply()` använder en funktion `FUN` på varje element i en lista `X`. Kör koden nedan: Vad har du skapat för lista?

```
myList <- split(x = ChickWeight, f = ChickWeight$Diet)
```

10. Räkna ut medelvärde, varians och percentiler för varje weight i varje element i `myList` med `lapply()`.
11. Skapa nu en funktion som kan ta ett dataset för en kyckling och räknar ut skillnaden i vikt mellan tidpunkt 0 och tidpunkt 10, om värden saknas för tidpunkt 0 eller 10 ska `NA` returneras. Använd denna funktion tillsammans med `lapply()` för att beräkna den skillnaden mellan tidpunkt 0 och 10 för alla kycklingar.

3 Minitillämpning: Fakultetsfunktion

Fakultet är definierat på följande sätt:

$$n! = \prod_{i=1}^{n} k$$

1. Skriv en funktion `MyFactorial()` med en for-loop som beräknar fakulteten för godtyckligt n.

2. Pröva din funktion med följande nummer $n = 1, 10, 100$.
3. Skriv en liknande funktion `MyFactorial2()` som istället använder en while-loop och pröva din funktion med följande nummer $n = 1, 10, 100$. Dubbelkolla att du får samma resultat som i uppgift 1) ovan.
4. Skriv en tredje funktion `MyFactorial3()` utan men med `prod()`.
5. Kör din funktion `MyFactorial(150)` 5000 gånger och undersök hur lång tid hela körningen tar med `system.time()`.
6. Upprepa uppgift ovan med `MyFactorial2()` och `MyFactorial3()`.
7. Vilket av de tre sätten att beräkna fakulteten är snabbast (effektivast)? Varför tror du?
8. Vad händer om du prövar `MyFactorial(500)`?
9. Notera att
$$\log(n!) = \sum_{i=1}^{n} \log(i)$$
Ändra din funktion `MyFactorial()` och lägg till ett extra argument `log` så att när `log = TRUE` returneras logaritmen av fakulteten och när `log = FALSE` returneras den vanliga fakulteten. Sätt värdet `TRUE` som defaultvärde.
10. Vad händer om du kör `MyFactorial(500)`? Om du kör `MyFactorial(500, log = FALSE)`?

Part II
Inlämningsuppgifter

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.

Ett förslag på hur du kan angripa problemet är att:

1. Lös uppgiften med vanlig kod direkt i R-Studio (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.

Varje uppgift kan ge flera poäng så även om du inte lyckas med alla delar i en funktion kan du få poäng.

4 Svenska personnummer

I Sverige har samtliga medborgare personnummer som de behåller livet ut och som används för identifikation. Personnummret består av tre delar, födelsedatum, födelsenummer och en kontrollsiffra. Som standard anges personnummer på följande sätt `ÅÅÅÅMMDDNNNK` där `ÅÅÅÅ` är födelseåret, `MM` födelsemånaden, `DD` födelsedagen, `NNN` födelsenumret och `K` kontrollsiffran.

Kontrollsiffran beräknas baserat på de övriga siffrorna i personnummret vilket gör att det är möjligt att kontrollera om ett personnummer är korrekt eller inte. Det är också möjligt att utifrån ett personnummer beräkna ålder och kön (samt för vissa även födelseort, men det spelar ingen roll i denna uppgift).

Detaljerna om för hur kön och kontrollsiffran beräknas finns i Skatteverkets broschyr SKV 704 [PDF]. Läs igenom denna broschyr innan du gör uppgiften nedan.

Exempel på personnummer som kan användas för att testa dina funktioner finns dels i broschyren från Skatteverket och dels på Wikipedia (sökord: “Personnummer i Sverige”). Du kan självklart även testa med ditt eget personnummer om du vill.

Syftet med denna uppgifter att skapa flera mindre funktioner och sedan kombinera ihop dessa funktioner till en större mer komplex funktion.

Det vi vill ha i slutändan är en funktion som tar en vektor med personnummer på olika format. Funktionen ska sedan returnera ett dataset med den information som finns i personnummret (med undantag för födelselän). Vi tar det dock i flera steg, med flera olika funktioner som utför olika steg. De stegen vi kommer göra är:

1. Skapa en funktion för att konvertera personnummer till ett standardformat som vi kan arbeta med vidare.
2. Skapa en funktion för att kontrollera kontrollsiffran i ett personnummer.
3. Skapa en funktion för att kontrollera om personnummret är ett samordningsnummer.
4. Skapa en funktion för att ta fram uppgift om kön från ett personnummer.
5. Kontrollera/skapa ett datum att beräkna ålder från.
6. Skapa en funktion för att ta fram uppgifter om ålder från ett personnummer och ett givet datum.
7. Skapa en funktion som sätter samman funktionerna ovan till en funktion, som tar en vektor av personnummer som input och returnerar ett dataset med personnummer och övrig information.

Följande funktioner kommer vara mycket användbara i denna uppgift: `paste()`, `substr()` och `Sys.Date()`. Kolla upp dessa funktioner innan du sätter igång.

4.1 Uppgift 1: `pnrFormat()`

Personnummer förekommer i många olika format i vanliga dataanalyser. De format funktionen ska kunna hantera är `ÅÅMMDD-NNNK`, `ÅÅMMDDNNNK` och `ÅÅÅÅMMDDNNNK`. Vi hoppar över personnummer på formen `ÅÅMMDD+NNNK`. Detta innebär att vi antar att alla personnummer är yngre än 100 år gamla. I R kan dessutom personnummer förekomma både som numeriska variabler faktorvariabler och som textvariabler. Vår funktion ska klara samtliga dessa fall.

Funktionen ska kunna ta ett personnummer på ett godtyckligt format och returnera personnummret som ett textelement med följande format: `ÅÅÅÅMMDDNNNK`

Ett förslag på de steg som kan ingå är:

1. Konvertera numeriska och faktorvariabler till text.
2. Använd en villkorssats för att hantera de tre olika formaten ovan [**Tips!** `nchar()`]

Här är textexempel på hur funktionen ska fungera:

```
pnr <- "640823-3234"
pnrFormat(pnr)

[1] "196408233234"

pnr <- 1311310324
pnrFormat(pnr)

[1] "201311310324"

pnr <- "198112189876"
pnrFormat(pnr)

[1] "198112189876"
```

4.2 Uppgift 2: `pnrCtrl()`

Nästa steg i funktionen är att kontrollera om ett personnummer är korrekt eller inte. För att beräkna en kontrollsiffra används den så kallade Luhn-algoritmen, mer information finns [här]. Vi ska skapa en funktion som använder Luhn-algoritmen för att testa om ett personnummer är korrekt eller inte. Fördelen nu är att vi vet exakt på vilket format personnummren kommer att vara eftersom vi kommer använda funktionen `pnrFormat()` innan vi anropar `pnrCtrl()`.

Funktionen ska ta argumentet `pnr` och returnera `TRUE` eller `FALSE` beroende på om personnummret är korrekt eller inte.

Ett förslag på hur funktionen kan implementeras är följande:

1. Dela upp personnummret så respektive siffra blir ett eget element. [**Tips!** `strsplit()` och `unlist()`]
2. Konvertera de uppdelade siffrorna till ett numeriskt format.
3. Den vektor av de enskilda siffrorna i personnummret kan nu användas i Luhn - algoritmen. Det enklaste sättet är att multiplicera personnummrets vektor med en beräkningsvektor av 0:or 1:or och 2:or på det sätt som beräkningen specificeras av Luhn-algoritmen.
   Obs! Skatteverkets beräkning görs inte på hela personnummret som returnerades av `pnrFormat()`, de delar som inte ska räknas kan sättas till 0 i beräkningsvektorn.
4. Nästa steg är att summera alla värden i vektorn ovan. Tänk på att tal större än 9 ska räknas som summan av tiotalssiffran och entalssiffran. [**Tips!** `%%` och `%/%`]
5. Summera värdena på vektorn som beräknades i 4 ovan. Plocka ut entalssiffran och dra denna entalssiffra från 10. Du har nu räknat ut kontrollsiffran. Puh!
6. Testa om den uträknade kontrollsiffran är samma som kontrollsiffran i personnummret.

Här är ett testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnrCtrl(pnr)

[1] TRUE

pnr <- "190101010101"
pnrCtrl(pnr)

[1] FALSE

pnr <- "198112189876"
pnrCtrl(pnr)

[1] TRUE

pnr <- "190303030303"
pnrCtrl(pnr)

[1] FALSE
```

4.3 Uppgift 3: `pnrSex()`

I denna uppgift ska vi från ett personnummer räkna ut det juridiska könet. Som framgår i skattebroschyren ska detta räknas ut genom att undersöka om den näst sista siffran i personnummret är jämt (kvinna) eller udda (man). Detta är vad som definierar en persons juridiska kön.

Skapa nu en funktion du kallar `pnrSex()`. Denna funktion ska ta ett personnummer och returnera en persons kön som ett textelement, `M` för man och `K` för kvinna.

Ett förslag på hur funktionen kan implementeras är följande:

1. Plocka ut den näst sista siffran i personnumret.
2. Konvertera denna siffra till numeriskt format och testa om siffran är jämn (returnera `K`) eller udda (returnera `M`)

Här är testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnrSex(pnr)

[1] "M"

pnr <- "190202020202"
pnrSex(pnr)

[1] "K"
```

4.4 Uppgift 4: `pnrSamordn()`

Vissa personer som inte är svenska medborgare kan få ett svenskt samordningsnummer som fungerar på samma sätt som personnummer. Då får man ett så kallad samordningsnummer. Den enda skillnaden är att att talet 60 har lagts till personnummrets födelsedatum (d.v.s. `DD` i `ÅÅÅÅMMDD`).

Skapa en funktion du kallar `pnrSamordn()` som tar ett personnummer på formatet genererat av funktionen `pnrFormat()` och returnerar `TRUE` om det är ett samordningsnummer och `FALSE` annars.

Ett förslag på hur funktionen kan implementeras är följande:

1. Plocka ut födelsedatumet ur personnummret.
2. Konvertera datumet till ett numeriskt värde och pröva om detta värde är större än 60.

Här är testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnrSamordn(pnr)

[1] FALSE

pnr <- "198112789876"
pnrSamordn(pnr)

[1] TRUE

pnr <- "198112189876"
pnrSamordn(pnr)

[1] FALSE
```

4.5 Uppgift 5: `pnrDate()`

Vi ska i denna funktion skapa ett datum för att senare beräkna åldern för olika individer. Skapa en funktion du kallar `pnrDate()` som tar argumentet `date`. Argumentet date ska ha följande textformat: `ÅÅÅÅ-MM-DD`. Om datumet inte är på detta format ska funktionen stoppas och returnera följande felmeddelande:

```
Incorrect date format:  Correct format should be YYYY-MM-DD.
```

Om inget datum anges av användaren ska den 31 december under föregående år returneras som datum.

Observera att senare i kursen kommer vi lära oss paketet `lubridate` som är betydligt bättre och enklare för att hantera och kontrollera datum och tid.

Ett förslag på hur funktionen kan implementeras är följande:

1. Ange ett defaultvärde för argumentet date som inte är aktuellt, ex. `NA`.
2. Om date har defaultvärdet, sätt datumvärdet den 31 december föregående år. [**Tips!** `is.na()`, `Sys.Date()` och `paste()`]
3. Testa om datumformatet är korrekt. Gör följande kontroller [**Tips!** `all()`]:
   (a) Är `ÅÅÅÅ`, `MM` och `DD` siffror. Detta kan kontrolleras genom att konvertera till numeriskt värde. Är det då inte siffror blir värdet `NA`. [**Tips!** `is.na()`]
   (b) Är `MM` större än 0 och mindre än 13.
   (c) Är `DD` större än 0 och mindre än 32.
4. Om datumformatet är är inkorrekt stoppa funktionen och returnera felmeddelandet ovan.
5. Annars, returnera datumet på korrekt format.

Här är testexempel på hur funktionen ska fungera:

```
pnrDate("2010-10-10")

[1] "2010-10-10"

pnrDate()

[1] "2013-12-31"

pnrDate("Hejbaberiba")

Warning:  NAs introduced by coercion
Warning:  NAs introduced by coercion
Warning:  NAs introduced by coercion
Incorrect date format:  Correct format should be YYYY-MM-DD.
```

4.6 Uppgift 6: `pnrAge()`

Sist ska vi baserat på dels ett personnummer och dels ett datum beräkna åldern för personen vid detta datum. Skapa en funktion du kallar `pnrAge()` tar argumentet `pnr` och argumentet `date`. Argumentet date ska ha följande textformat: `ÅÅÅÅ-MM-DD`. Du kan utgå från att formatet är på detta sätt då `pnrDate()` kommer anropas innan denna funktion.

Ett förslag på hur funktionen kan implementeras är följande:

1. Räkna ut skillnaden i hela år mellan datumets årtal personnummrets årtal. D.v.s hur gammal personen är den 31 december.
2. Pröva om månad och dag är större (senare) för `pnr` än för `date`. Om så är fallet dra av ett år från årtalsberäkningen ovan (personen har ännu inte fyllt år) och returnera åldern vid det givna datumet.

Här är testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnrAge(pnr, date = "2010-10-10")

[1] 46

pnr <- "198112189876"
pnrAge(pnr, date = "2014-12-31")

[1] 33
```

4.7 Uppgift 7: `pnrInfo()`

Nu har vi skapat ett antal funktioner för att beräkna olika delar av personnummret. Nu ska vi sätta ihop dessa funktioner till en enda funktion som baserat på en vektor av personnummer returnerar en `data.frame` följande variabler:

1. `pnr`: personnummret i textformat,
2. `correct`: information om personnummret är korrekt,
3. `samordn`: om personnummret är ett samordningsnummer
4. `sex`: kön och
5. `age`: ålder i år

Det ska också vara möjligt att skicka vidare datum till funktionen `pnrDate()`, men om inget skickas med ska defaultvärdet i `pnrDate()` användas.

Funktionen ska dessutom generera följande meddelande med `message()`:

```
The age has been calculated at [DATUM].
```

Ett förslag på hur funktionen kan implementeras är följande:

1. Kontrollera/skapa ett korrekt datum för att beräkna ålder med `pnrDate()`.
2. Formatera om alla personnummer i inputvektorn till standardformatet med `pnrFormat()`.
3. Räkna ut vilka personnummer som är korrekta personnummer med `pnrCtrl()`.
4. Använd `pnrSamordn()` för att skapa en vektor över vilka personnummer som är samordningsnummer.
5. Använd `pnrSex()` för att räkna ut könet för respektive personnummer.
6. Dra av 6 från första siffran i födelsedatumet för de personnummer som är samordningsnummer. Du kan antingen göra detta direkt i `pnrInfo()` eller skapa en till egen funktion som gör just detta.

7. Använd `pnrAge()` för att beräkna åldern för respektive personnummer.
8. Skriv ut meddelandet ovan med `message()`.
9. Sätt samman dessa resultat till den `data.frame` som ska returneras.

Här är testexempel på hur funktionen ska fungera:

```
pnr <- c("196408233234", "640883-3234", "198112189876")
pnrInfo(pnr)

The age has been calculated at 2013-12-31.

           pnr correct samordn sex age
1 196408233234    TRUE   FALSE   M  49
2 196408833234   FALSE    TRUE   M  49
3 198112189876    TRUE   FALSE   M  32

pnrInfo(pnr, date = "2000-06-01")

The age has been calculated at 2000-06-01.

           pnr correct samordn sex age
1 196408233234    TRUE   FALSE   M  35
2 196408833234   FALSE    TRUE   M  35
3 198112189876    TRUE   FALSE   M  18
```

4.8 Tävlingsuppgift (ej obligatorisk): Optimera `pnrInfo()`

För de studenter som vill och är intresserade kan pröva att optimera funktionen `pnrInfo()` för så snabb beräkningshastighet som möjligt. Generellt sett ska optimering av kod endast göras efter att den koden fungerar som den ska. Eller som en av datorvetenskapens giganter, **Donald Knuth**, har uttryckt det:

> Premature optimization is the root of all evil.

Om du vill delta i tävlingen skapar du en ny R - fil med ditt slutliga förslag `pnrInfoSnabbast()`. Kontrollera att den fungerar med testexemplet. Vinnaren är den med den snabbaste **fungerande** funktionen.
Obs! Du måste lämna in den vanliga labben också.
Generellt i R kan man tänka på följande om man vill optimera beräkningshastigheten för en funktion:

1. `for` loopar är långsammare än `*apply`-funktioner.
2. det går snabbare om man först skapar en datastruktur och sedan fyller den med värden än att skapa en ny datastruktur i varje iteration.

Det finns säkert en massa andra sätt! Pröva och testa hur du effektivisera koden med `system.time()`. Här är ett exempel på hur ni kan pröva hur snabba era funktioner är:

```
testSpeed <- sample(c("196408233234", "640883-3234", "198112189876"), size = 10000,
    replace = TRUE)
system.time(pnrInfo(testSpeed))

The age has been calculated at 2013-12-31.

   user  system elapsed
  1.649   0.007   1.658
```

Nu är du klar!